UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ELITON LIMA ROSENDO FILHO

AVALIAÇÃO DE PLATAFORMAS DE CONTÊINERES AWS PARA *OFFLOADING* COMPUTACIONAL DE APRENDIZADO DE MÁQUINA EM DISPOSITIVOS DE BAIXO PODER COMPUTACIONAL

QUIXADÁ

2025

ELITON LIMA ROSENDO FILHO

AVALIAÇÃO DE PLATAFORMAS DE CONTÊINERES AWS PARA *OFFLOADING* COMPUTACIONAL DE APRENDIZADO DE MÁQUINA EM DISPOSITIVOS DE BAIXO PODER COMPUTACIONAL

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Orientador: Prof. Dr. João Marcelo Uchôa de Alencar.

QUIXADÁ

2025

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

R725a Rosendo Filho, Eliton Lima.
Avaliação de plataformas de contêineres AWS para offloading computacional de aprendizado de máquina em dispositivos de baixo poder computacional / Eliton Lima Rosendo Filho. – 2025.
53 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Ciência da Computação, Quixadá, 2025.
Orientação: Profa. Dra. João Marcelo Uchôa de Alencar.

1. computational offloading. 2. computação em nuvem. 3. aws fargate. 4. aws ec2. 5. contêineres. I. Título.

CDD 004

ELITON LIMA ROSENDO FILHO

AVALIAÇÃO DE PLATAFORMAS DE CONTÊINERES AWS PARA *OFFLOADING* COMPUTACIONAL DE APRENDIZADO DE MÁQUINA EM DISPOSITIVOS DE BAIXO PODER COMPUTACIONAL

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Aprovada em: 29/07/2025.

BANCA EXAMINADORA

Prof. Dr. João Marcelo Uchôa de Alencar (Orientador)
Universidade Federal do Ceará (UFC)

Regis Pires Magalhães
Universidade Federal do Ceará (UFC)

Prof. Dr. Bruno Góis Mateus
Universidade Federal do Ceará - UFC

RESUMO

A crescente demanda por aplicações de Aprendizado de Máquina (ML) em dispositivos com poder computacional limitado impõe desafios significativos de desempenho e consumo de energia. Este trabalho aborda essa problemática através do *offloading* computacional, avaliando a viabilidade e a eficiência de plataformas de contêineres na nuvem para executar tarefas de inferência de ML. O objetivo principal é comparar duas abordagens da Amazon Web Services (AWS): o serviço sem servidor AWS ECS Fargate e a execução de contêineres em instâncias gerenciadas AWS ECS com EC2. Para a análise, desenvolve-se uma aplicação protótipo de classificação de imagem, utilizando o modelo MobileNetV3Large, e realizam-se experimentos sistemáticos para coletar métricas de latência, tempo total de processamento e custo financeiro. Os resultados demonstram que, embora a abordagem com EC2 apresente um desempenho marginalmente superior em termos de latência, o Fargate se destaca como a opção mais econômica e com menor complexidade operacional. Conclui-se que o Fargate representa uma solução robusta e financeiramente viável para cenários de *offloading* de ML, oferecendo um trade-off vantajoso entre custo, simplicidade e um desempenho adequado para a maioria das aplicações.

Palavras-chave: offloading computacional; computação em nuvem; aws fargate; aws ec2; contêineres.

ABSTRACT

The growing demand for Machine Learning (ML) applications on devices with limited computing power poses significant performance and energy consumption challenges. This work addresses this issue through computational offloading, evaluating the feasibility and efficiency of cloud container platforms for executing ML inference tasks. The main objective is to compare two Amazon Web Services (AWS) approaches: the serverless AWS ECS Fargate service and running containers on AWS ECS managed instances with EC2. For the analysis, a prototype image classification application is developed using the MobileNetV3Large model, and systematic experiments are conducted to collect metrics on latency, total processing time, and financial cost. The results demonstrate that, although the EC2 approach presents marginally superior performance in terms of latency, Fargate stands out as the more cost-effective option with lower operational complexity. It is concluded that Fargate represents a robust and financially viable solution for ML offloading scenarios, offering an advantageous trade-off between cost, simplicity, and adequate performance for most applications.

Keywords: computational offloading; cloud computing; aws fargate; aws ec2; containers.

LISTA DE FIGURAS

Figura 1 – Arquitetura Docker 17
Figura 2 – Infraestrutura como serviço *Serverless* 20
Figura 3 – Arquitetura *Fargate* 23
Figura 4 – Arquitetura *Offloading* Computacional 26
Figura 5 – Fluxograma Metodologia. 35
Figura 6 – Diagrama de Arquitetura. 39
Figura 7 – Comparativo Tempo Total Médio. 44
Figura 8 – Comparativo Tempo Médio por Imagem. 45
Figura 9 – Comparativo Custo Médio. 46

LISTA DE TABELAS

Tabela 1 – Comparativo Detalhado de Desempenho e Custo entre AWS Fargate e EC2 . 43
Tabela 2 – Comparativo Consolidado de Desempenho e Custo Médio entre AWS Fargate e EC2 43

LISTA DE QUADROS

Quadro 1 – Comparativo de Aspectos Conceptuais e de Aplicação dos Trabalhos Relacionados 33
Quadro 2 – Comparativo de Aspectos de Implementação e Infraestrutura dos Trabalhos Relacionados 33
Quadro 3 – Comparativo de Trabalhos por Abordagem Metodológica e Métricas 33

SUMÁRIO

1 INTRODUÇÃO 10
1.1 Objetivos 11
1.1.1 Objetivo geral 11
1.1.2 Objetivos Específicos 11
1.2 Estrutura do Trabalho 12
2 FUNDAMENTAÇÃO TEÓRICA 13
2.1 Aprendizado de Máquina 13
2.2 Análise de Imagem 14
2.2.1 Classificação de Imagem 15
2.3 Conteinerização de Aplicações e Docker 15
2.4 Python 17
2.4.1 Tensor Flow 18
2.4.2 MobileNetV3 18
2.4.3 Flask 19
2.5 ***Terraform*** 19
2.6 Computação em Nuvem 19
2.7 ***Amazon Web Services*** 21
2.7.1 EC2 21
2.8 ***Elastic Container Registry*** 22
2.8.1 Elastic Container Service 22
2.8.1.1 Fargate *23*
2.8.1.2 Instância EC2 . . . . . *24*
2.9 Computação sem servidor 24
2.10 ***Offloading*** **Computacional** 25
2.11 Métricas de Avaliação 27
3 TRABALHOS RELACIONADOS 29
3.1 Acelerando a identificação de impressões digitais através da técnica do ***offloading*** **computacional** 29
3.2 Avaliação das Linguagens Python e Go na Plataforma AWS Lambda para Microsserviços de Computação de Alto Desempenho 30

3.3 **Análise Comparativa de Desempenho e Custo em Serviços de Contêineres Sem Servidor** 31
3.4 ***Modernizing Applications with Containers in Public Cloud*** 32
3.5 **Análise Comparativa** 32
4 **METODOLOGIA** 35
4.1 **Definição de Volume de imagens e métricas de avaliação** 35
4.2 **Definição dos Serviços AWS** 36
4.3 **Desenvolvimento de um protótipo de Aplicação com Modelos Pré-Treinados** 37
4.4 **Arquitetura de Rede para *Offloading* Computacional** 39
4.5 **Aplicação do *Offloading* Computacional** 41
4.6 **Testes e Validação** 42
4.7 **Comparação de Desempenho e Custo** 42
5 **RESULTADOS** 43
5.1 **Apresentação dos Dados Experimentais** 43
5.2 **Gráficos Comparativos** 44
5.3 **Considerações Finais da Seção** 46
6 **CONCLUSÕES E TRABALHOS FUTUROS** 48
6.1 **Conclusão** 48
6.2 **Trabalhos Futuros** 49
REFERÊNCIAS 51

1 INTRODUÇÃO

A crescente demanda por aplicações inteligentes em dispositivos com poder computacional limitado impõe um desafio significativo, especialmente quando essas aplicações executam tarefas que exigem alto processamento (Santos *et al.*, 2016). Para superar essa limitação, o *offloading* computacional surge como uma técnica crucial, permitindo que operações intensivas sejam delegadas a servidores remotos mais robustos(Barbosa *et al.*, 2021), aliviando a carga nos dispositivos locais e aproveitando a capacidade das infraestruturas de nuvem (Flores *et al.*, 2015). Essa abordagem é particularmente vital em cenários onde a eficiência energética e o desempenho são críticos, como em dispositivos móveis, garantindo a execução de aplicações complexas sem comprometer a experiência do usuário (Yao *et al.*, 2020).

A computação em nuvem desempenha um papel fundamental ao suportar essas arquiteturas de *offloading*. O poder de processamento acessível via nuvem é o principal viabilizador para a execução remota de tarefas pesadas (Silva *et al.*, 2020). A transição de dados e processamento de sistemas locais para *data centers* remotos tem permitido que empresas acessem recursos computacionais escaláveis sem a necessidade de grandes investimentos em *hardware* (Silva, 2016).

A nuvem oferece escalabilidade sem precedentes, na qual os recursos podem ser ajustados com base na demanda, otimizando custos operacionais e evitando desperdícios de capacidade (Silva *et al.*, 2020). Essa flexibilidade é crucial para aplicações que exigem poder computacional variável. Além disso, a computação em nuvem permite a utilização de plataformas de execução de contêineres(Chen, 2019), que se destacam por sua eficiência e portabilidade, sendo ideais para o empacotamento e implantação dessas aplicações.

O campo de *Machine Learning* (ML), ou Aprendizado de Máquinas, é uma das áreas que mais se beneficia desta abordagem. ML é uma subárea da inteligência artificial que se concentra no desenvolvimento de algoritmos capazes de aprender a partir de dados para realizar previsões ou tomar decisões (Ludermir, 2021; Baranauskas; Monard, 2000). Muitas de suas aplicações, no entanto, são computacionalmente intensivas.

Uma das aplicações mais promissoras e exigentes de Aprendizado de Máquina é a classificação de imagens. Este campo envolve o uso de algoritmos para entender e interpretar dados visuais de forma semelhante à cognição humana (Costa; Couto, 2008), sendo amplamente utilizado em tarefas complexas nas áreas de segurança e saúde (Elyan *et al.*, 2022). Um exemplo notável é o sistema de reconhecimento facial DeepFace, que demonstra o poder do ML na

transformação de dados brutos em conhecimento acionável (Awari, 2024; Ludermir, 2021).

O presente trabalho tem como objetivo principal avaliar a viabilidade de uma solução para o *offloading* computacional de aplicações de classificação de imagem utilizando Aprendizado de Máquina a partir de dispositivos de baixa capacidade computacional. Para isso, a pesquisa propõe a utilização de plataformas de execução de contêineres na nuvem, permitindo que o processamento das inferências seja delegado a infraestruturas escaláveis. Esta estratégia visa mitigar as restrições de *hardware* dos dispositivos locais, garantindo maior eficiência no uso de recursos.

Para alcançar este objetivo, investiga-se e compara-se dois serviços de execução de contêineres da AWS. A pesquisa avalia o impacto do uso dessas infraestruturas para a execução remota de inferências de modelos pré-treinados, como o MobileNetV3 (Howard *et al.*, 2019). conduzem-se experimentos para medir a latência individual, a vazão total e a viabilidade econômica de cada abordagem, oferecendo *insights* sobre a aplicabilidade dessas plataformas para o processamento remoto de Aprendizado de Máquina.

1.1 Objetivos

Esta seção é referente ao objetivo geral e específicos do presente trabalho.

1.1.1 Objetivo geral

O objetivo deste trabalho é avaliar uma solução que permita a execução de aplicações de classificação de imagem utilizando Aprendizado de Máquina em dispositivos com capacidade computacional limitada, baseando-se em tecnologias de plataformas de execução de contêineres (como AWS ECS Fargate e EC2) em conjunto com técnicas de *offloading* computacional e inferência, oferecendo uma alternativa eficiente para contornar as restrições de *hardware* desses dispositivos.

1.1.2 Objetivos Específicos

- Prototipação de uma aplicação de classificação de imagem utilizando bibliotecas com modelos pré-treinados.
- Projetar e avaliar uma arquitetura baseada em *offloading* computacional computacional utilizando Amazon ECS Fargate e ECS instâncias EC2.

- Realizar experimentos para medir latência e custo das execuções em diferentes infraestruturas.
- Validar a escalabilidade e eficiência das soluções propostas.

1.2 Estrutura do Trabalho

Este documento está estruturado da seguinte forma: o Capítulo 2 apresenta a fundamentação teórica dos conceitos e tecnologias essenciais. O Capítulo 3 discute os trabalhos relacionados, contextualizando a pesquisa. O Capítulo 4 detalha a metodologia empregada, enquanto o Capítulo 5 apresenta e analisa os resultados obtidos. Finalmente, o Capítulo 6 expõe as conclusões do trabalho e sugere direções para futuras pesquisas.

2 FUNDAMENTAÇÃO TEÓRICA

A seção de fundamentação teórica deste trabalho visa estabelecer as bases conceituais e metodológicas que sustentam a pesquisa desenvolvida. Nesta etapa, onde se explora os principais conceitos e tecnologias envolvidos, como Aprendizado de Máquina, classificação de imagens, computação em nuvem, computação sem servidor e *offloading* computacional. O objetivo é oferecer uma visão abrangente e detalhada desses elementos, evidenciando como eles se inter-relacionam e contribuem para a solução proposta no contexto deste projeto.

2.1 Aprendizado de Máquina

O campo de Aprendizado de Máquina tem-se tornado cada vez mais relevante, especialmente no contexto da classificação de imagens, um dos ramos mais dinâmicos e promissores da inteligência artificial (IA) (Goodfellow *et al.*, 2016). Segundo Baranauskas e Monard (2000), ML é uma subárea da IA focada no desenvolvimento de algoritmos e modelos capazes de aprender a partir de dados, permitindo que sistemas façam previsões ou tomem decisões sem a necessidade de serem explicitamente programados para cada tarefa específica.

O Aprendizado de Máquina baseia-se na ideia de que os sistemas podem melhorar automaticamente sua performance ao serem expostos a mais dados, sem a necessidade de serem programados explicitamente IBM (2024).

Isso é particularmente relevante na classificação de imagens, onde grandes volumes de dados visuais precisam ser processados para identificar padrões, classificar objetos, ou até mesmo detectar anomalias. O processo de aprendizado envolve a construção de modelos que são treinados com um conjunto de dados conhecido, permitindo que o sistema aprenda a associar entradas específicas a saídas desejadas.

Existem várias abordagens em ML que podem ser aplicadas à análise de imagens, incluindo:

- Aprendizado Supervisionado: De acordo com Li (2018) este método envolve a utilização de um conjunto de dados rotulado, onde o sistema aprende a mapear entradas (como imagens) para saídas (como categorias de objetos) com base em exemplos pré-definidos. Por exemplo, na classificação de imagens, o modelo é treinado para reconhecer e categorizar imagens de gatos, carros, árvores, etc.
- Aprendizado Não Supervisionado: Ao contrário do aprendizado supervisionado, este

método não utiliza rótulos explícitos, o sistema tenta identificar padrões ou agrupamentos nos dados, o que é útil em tarefas como segmentação de imagens, onde o objetivo pode ser identificar regiões semelhantes dentro de uma imagem, Li (2018) destaca que o aprendizado não supervisionado tenta extrair informações de dados sem rótulos.

– Aprendizado por Reforço: Segundo Li (2018) o aprendizado por reforço é sequencial, de longo prazo, considerando a recompensa acumulada ao longo do tempo. Esta abordagem é baseada na ideia de que um agente aprende a tomar decisões por meio de recompensas ou punições. Embora menos comum na análise de imagens tradicionais, pode ser útil em aplicações que envolvem a navegação ou interação com ambientes visuais, como em robótica.

2.2 Análise de Imagem

O reconhecimento de imagem em Aprendizado de Máquina é um campo de estudo dinâmico e de grande relevância, que combina algoritmos sofisticados e técnicas avançadas para capacitar as máquinas a entender, analisar e interpretar imagens de forma semelhante à cognição humana(Krizhevsky *et al.*, 2012). Segundo Awari (2024), esse campo está em constante evolução, impulsionado por avanços tecnológicos e pela crescente demanda por soluções automatizadas em diversas indústrias.

A análise de imagem, como destacado por Ludermir (2021), é uma ferramenta poderosa aplicada em uma ampla gama de áreas, incluindo segurança, saúde, automação industrial e entretenimento. Um exemplo emblemático é o sistema de reconhecimento facial DeepFace, desenvolvido pelo Facebook, que utiliza redes neurais profundas para identificar rostos em imagens com uma precisão impressionante. Esse tipo de sistema demonstra como a análise de imagem pode reconhecer e classificar informações visuais de maneira eficiente, oferecendo benefícios significativos em termos de automação e segurança.

Em suma, o reconhecimento de imagens em Aprendizado de Máquina combina métodos supervisionados e não supervisionados para abordar uma variedade de desafios na interpretação de dados visuais. A escolha do método depende da disponibilidade de rótulos nos dados e do objetivo específico da aplicação, seja para a construção de classificadores robustos ou para a descoberta de padrões ocultos. O contínuo avanço nesse campo promete trazer ainda mais precisão e eficiência às aplicações que dependem da interpretação automatizada de imagens.

2.2.1 Classificação de Imagem

A classificação de imagem é uma tarefa fundamental no campo da visão computacional e do Aprendizado de Máquina, que envolve a atribuição de um rótulo ou categoria predefinida a uma imagem inteira. O objetivo é que um sistema computacional seja capaz de identificar o objeto ou a cena principal contida em uma imagem e classificá-la corretamente entre um conjunto de classes possíveis (ex: "gato", "cachorro", "carro", "árvore") (Russakovsky *et al.*, 2015).

Essa tarefa é tipicamente resolvida por meio de Redes Neurais Convolucionais (CNNs), que são arquiteturas de Aprendizado Profundo projetadas para processar dados de pixel com uma estrutura de grade, como imagens. As CNNs aprendem automaticamente características hierárquicas das imagens, desde bordas e texturas em camadas iniciais até partes de objetos e objetos completos em camadas mais profundas (Goodfellow *et al.*, 2016).

O processo de classificação de imagem geralmente segue as seguintes etapas:

1. **Entrada da Imagem:** A imagem é fornecida ao modelo em formato de pixel.
2. **Extração de Características:** Camadas convolucionais da CNN aplicam filtros para extrair características relevantes.
3. **Classificação:** As características extraídas são passadas para camadas totalmente conectadas, que realizam a classificação, resultando em probabilidades para cada categoria possível.
4. **Saída:** A categoria com a maior probabilidade é atribuída à imagem como sua classe predita.

A classificação de imagem possui vasta aplicação em diversas áreas, incluindo veículos autônomos (reconhecimento de sinais de trânsito), medicina (diagnóstico assistido por imagem), segurança (detecção de anomalias), e comércio eletrônico (organização de produtos) (Sankaranarayanan; Kavitha, 2020). No contexto deste trabalho, a classificação de imagem é a tarefa central de Aprendizado de Máquina a ser descarregada (*offloaded*) para a nuvem, visando otimizar o processamento em dispositivos de baixa capacidade computacional.

2.3 Conteinerização de Aplicações e Docker

A conteinerização de aplicações refere-se a uma tecnologia de virtualização a nível de sistema operacional que empacota o código de uma aplicação, suas dependências, bibliotecas e configurações em uma unidade isolada e padronizada conhecida como contêiner. Diferentemente das máquinas virtuais (VMs), que virtualizam o *hardware* completo e requerem um sistema

operacional (*guest OS*) para cada instância, os contêineres compartilham o *kernel* do sistema operacional (*host OS*) subjacente, o que os torna significativamente mais leves, mais rápidos para iniciar e mais eficientes em termos de uso de recursos (Chen, 2019).

As principais vantagens da contenção incluem:

- **Portabilidade:** Um contêiner pode ser executado consistentemente em qualquer ambiente que possua um ambiente de execução de contêiner compatível, garantindo que a aplicação se comporte da mesma forma em desenvolvimento, teste e produção.
- **Isolamento:** Os contêineres fornecem um ambiente isolado para cada aplicação, impedindo conflitos de dependências entre diferentes aplicações e garantindo que os recursos alocados para uma aplicação não interfiram em outras.
- **Eficiência de Recursos:** Ao compartilhar o *kernel* do sistema operacional e iniciar mais rapidamente que as VMs, os contêineres permitem uma maior densidade de aplicações por servidor e um uso mais otimizado do *hardware* subjacente.
- **Escalabilidade:** A leveza e o rápido tempo de inicialização dos contêineres facilitam a escalabilidade horizontal de aplicações, permitindo que novas instâncias sejam criadas ou removidas rapidamente em resposta à demanda.

Dentro do ecossistema de contenção, o Docker surgiu como a plataforma de código aberto mais amplamente adotada e que revolucionou o desenvolvimento e a implantação de software a partir de 2013 (Red Hat, 2025; DIO - Digital Innovation One, 2025) . O Docker facilita a criação, o empacotamento, a distribuição e a execução de contêineres, tornando a tecnologia acessível a desenvolvedores e equipes de operação.

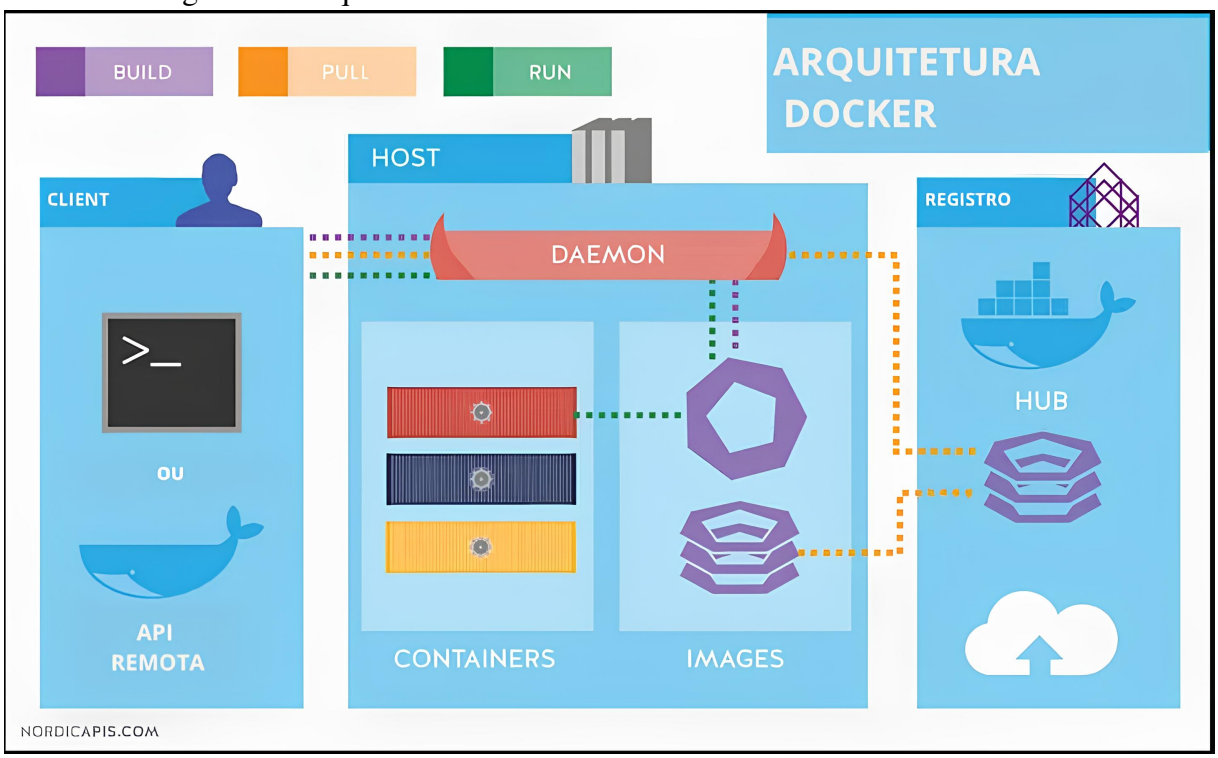
A arquitetura do Docker (conforme ilustrado na Figura 1) é baseada em um modelo cliente-servidor e consiste em componentes-chave:

- **Docker Engine:** O daemon (processo em segundo plano) que constrói e executa contêineres.
- **Docker CLI (Command Line Interface):** A ferramenta de linha de comando que permite aos usuários interagir com o Docker Engine.
- **Docker Image:** Um pacote executável, autocontido e imutável que inclui tudo o que é necessário para executar uma aplicação, incluindo o código, o *runtime*, bibliotecas e configurações.
- **Docker Container:** Uma instância executável de uma imagem Docker. É o ambiente isolado onde a aplicação roda.
- **Docker Registry (e.g., Docker Hub, Amazon ECR):** Um serviço para armazenar e distribuir

imagens Docker.

A capacidade do Docker de encapsular software e suas dependências em unidades portáteis e isoladas resolveu um dos maiores desafios do ciclo de vida de software – a compatibilidade de ambiente (Chen, 2019). Isso tem sido fundamental para a adoção de práticas modernas de DevOps e para o desenvolvimento de arquiteturas de microsserviços (Red Hat, 2025), onde a contenção desempenha um papel central para a agilidade e a escalabilidade de aplicações na nuvem.

Figura 1 – Arquitetura Docker



Fonte: Nordicaps

2.4 Python

Python é uma linguagem de programação de alto nível, interpretada e de propósito geral, conhecida por sua simplicidade e legibilidade. Criada por Guido van Rossum em 1991, Python foi projetada com foco em produtividade e facilidade de uso, utilizando uma sintaxe clara que permite aos desenvolvedores concentrar-se na lógica da aplicação em vez de detalhes de implementação (Borges, 2014). Seu ecossistema robusto, com uma ampla variedade de bibliotecas e *frameworks*, torna Python uma escolha ideal para aplicações em áreas como desenvolvimento web, ciência de dados, inteligência artificial e automação. A filosofia "*batteries*

included"contribui para sua versatilidade, disponibilizando ferramentas integradas para resolver problemas complexos de forma eficiente.

2.4.1 Tensor Flow

TensorFlow é um framework open-source desenvolvido pelo Google que permite a criação e treinamento de modelos de Aprendizado de Máquina e redes neurais profundas. Projetado para ser escalável e flexível, ele suporta execução em CPUs, GPUs e TPUs, tornando-o ideal para aplicações de alto desempenho em inteligência artificial (Pang *et al.*, 2020). Com sua API de alto nível, o TensorFlow facilita a prototipagem rápida, enquanto suas ferramentas avançadas, como TensorBoard, oferecem visualizações interativas que ajudam a compreender o comportamento dos modelos. Amplamente utilizado em indústrias como saúde, finanças e tecnologia, o TensorFlow desempenha um papel crucial na democratização do Aprendizado de Máquina(Pang *et al.*, 2020).

2.4.2 MobileNetV3

O MobileNetV3 representa a evolução da arquitetura MobileNet, projetado especificamente para otimizar o desempenho em CPUs de dispositivos móveis. Este modelo foi desenvolvido por meio de uma combinação de pesquisa de arquitetura de rede (*Neural Architecture Search* – NAS) com reconhecimento de hardware, complementada pelo algoritmo NetAdapt, e aprimoramentos arquitetônicos inovadores (Howard *et al.*, 2019).

A arquitetura MobileNetV3 foi otimizada para oferecer um balanceamento ainda mais eficaz entre precisão e latência, superando seu antecessore MobileNetV2. Existem duas variantes principais:

- **MobileNetV3 Small:** Projetado para cenários com recursos mais restritos, apresenta mais precisão na classificação de imagens no conjunto de dados ImageNet com latência similar ao MobileNetV2.
- **MobileNetV3 Large:** Desenvolvido para um desempenho superior, atinge mais precisão no ImageNet e consegue uma redução de latência em comparação com o MobileNetV2.

Para alcançar essas melhorias, o MobileNetV3 incorpora novas técnicas, como a introdução do bloco *Squeeze-and-Excitation* e o uso da função de ativação *h-swish*, que são otimizadas para eficiência computacional em ambientes de baixa capacidade (Howard *et al.*, 2019).

2.4.3 Flask

Flask é um *microframework* Python projetado para criar aplicações web leves e rápidas. Simples, mas poderoso, ele permite aos desenvolvedores construir APIs e serviços web com facilidade, utilizando uma arquitetura modular que pode ser ampliada por meio de extensões (Grinberg, 2018). Apesar de sua simplicidade, Flask é altamente configurável, sendo amplamente utilizado em projetos que variam de protótipos a sistemas de produção complexos (Grinberg, 2018). Sua flexibilidade e suporte a padrões modernos da web, como REST, tornam-no uma escolha popular para desenvolvedores que precisam de controle detalhado sobre suas aplicações.

2.5 *Terraform*

Terraform é uma ferramenta *open-source* desenvolvida pela HashiCorp, projetada para implementar e gerenciar infraestruturas de TI de maneira automatizada, utilizando o conceito de Infraestrutura como Código (IaC) (Brikman, 2022). Por meio de arquivos de configuração declarativos, é possível descrever recursos de infraestrutura, como servidores, bancos de dados, redes e muito mais, que podem ser provisionados em diversos provedores de nuvem, incluindo AWS, Azure e Google Cloud (Jayaram *et al.*, 2024). Essa abordagem não apenas garante a consistência entre ambientes, mas também facilita a replicação e modificação de recursos de forma eficiente. Além disso, o Terraform possui suporte a módulos reutilizáveis, permitindo maior organização e reutilização de código.

O Terraform revolucionou a maneira como organizações gerenciam seus ambientes de TI, proporcionando automação, versionamento e rastreabilidade. Uma das suas funcionalidades mais poderosas é o plano de execução, que permite prever mudanças na infraestrutura antes de serem aplicadas, garantindo maior controle e segurança operacional (Brikman, 2022). Sua integração com ferramentas de CI/CD (Integração Contínua/Entrega Contínua) também facilita o desenvolvimento ágil, possibilitando atualizações contínuas e minimizando erros manuais. Essa eficiência e flexibilidade fazem do Terraform uma escolha indispensável para equipes que buscam modernizar e otimizar a gestão de infraestrutura em ambientes *multicloud* e *on-premises*.

2.6 Computação em Nuvem

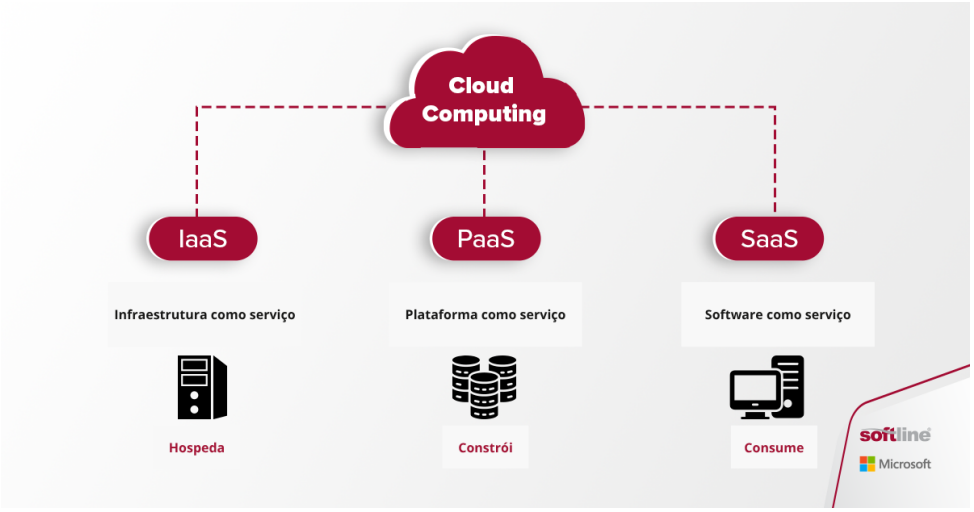
A computação em nuvem revolucionou a maneira como armazenamos e processamos dados, marcando uma transição significativa do armazenamento local em computadores indi-

viduais e infraestruturas de TI institucionais para sistemas baseados em *datacenters* remotos e extensivos, os mesmos são operados por provedores de serviços que oferecem armazenamento e capacidades computacionais como serviços pagos, facilitando o acesso a recursos computacionais escaláveis sem o investimento inicial em *hardware* caro Silva *et al.* (2020).

A essência da computação em nuvem reside na sua capacidade de descentralizar o armazenamento de dados e processamento. Isso não só melhora a eficiência pela distribuição geográfica que reduz a latência para diversos usuários, mas também permite uma escalabilidade sem precedentes(Mell; Grance, 2011). As empresas podem expandir ou reduzir seus recursos computacionais com base na demanda, otimizando custos operacionais e evitando o desperdício de capacidade (Silva, 2016).

Segundo a Microsoft (2024) (Figura 2). Os modelos de serviço em nuvem incluem Infraestrutura como Serviço (IaaS), Plataforma como Serviço (PaaS) e Software como Serviço (SaaS), cada um oferecendo diferentes níveis de controle, flexibilidade e gestão. IaaS permite aos usuários alugar infraestrutura física e virtual, PaaS fornece plataformas de desenvolvimento e execução sem a complexidade de gerenciar o *hardware* subjacente, e SaaS entrega aplicações completas como um serviço.

Figura 2 – Infraestrutura como serviço *Serverless*



Fonte: Microsoft (2024)

Os benefícios da computação em nuvem incluem redução de custos, acesso a tecnologias avançadas, alta disponibilidade e recuperação de desastres melhorada(Carr, 2009). No entanto, desafios como segurança dos dados, conformidade regulatória e dependência de conexões de internet são preocupações que ainda necessitam de atenção contínua.

A adoção da computação em nuvem transformou modelos de negócio, possibilitando

que *startups* e empresas estabelecidas inovem rapidamente, o modelo de pagamento conforme o uso também permite que organizações de todos os tamanhos acessem recursos que anteriormente eram proibitivos, democratizando o acesso à tecnologia de ponta (Carr, 2009).

2.7 *Amazon Web Services*

Segundo a Amazon Web Services (Amazon Web Services, 2025b) é amplamente reconhecida como uma das principais plataformas de computação em nuvem do mundo, oferecendo uma vasta gama de serviços que suportam desde startups até grandes corporações e governos. Com sua infraestrutura global altamente escalável e resiliente, AWS permite que empresas criem, implementem e gerenciem aplicações com eficiência, eliminando a necessidade de investir em *hardware* físico. Entre os serviços mais utilizados estão o Amazon S3 para armazenamento de objetos, o EC2 para instâncias de computação, e o RDS para bancos de dados relacionais gerenciados. Essa flexibilidade é complementada por ferramentas avançadas de inteligência artificial, Aprendizado de Máquina, *big data* e monitoramento, como o Amazon SageMaker e o CloudWatch, que possibilitam inovações rápidas e tomadas de decisão baseadas em dados(Amazon Web Services, 2024).

AWS tem desempenhado um papel crucial na transformação digital, ajudando a democratizar o acesso à tecnologia de ponta (Amazon Web Services, 2024). De acordo com AWS sua abordagem é baseada em pay-as-you-go (pague pelo que usar) revolucionou a forma como recursos tecnológicos são consumidos, permitindo que até mesmo pequenas organizações acessem soluções que anteriormente eram exclusivas de grandes empresas com infraestrutura robusta (Amazon Web Services, 2024). A maturidade e robustez dos serviços da AWS, combinadas com seu compromisso com segurança e conformidade, consolidam sua posição como um parceiro confiável em setores críticos, incluindo saúde, finanças e ciência. A constante inovação da AWS tem garantido que ela permaneça à frente das demandas de um mundo cada vez mais conectado e dependente de tecnologia (Amazon Web Services, 2024).

2.7.1 EC2

O EC2 (*Elastic Compute Cloud*) (Barbosa; Rego, 2019) é um serviço da AWS que oferece capacidade de computação escalável na nuvem. Ele permite que desenvolvedores implementem e gerenciem instâncias virtuais de servidores, configuradas para atender às necessidades

específicas de suas aplicações. O EC2 oferece uma variedade de tipos de instâncias, otimizados para diferentes cargas de trabalho, como computação intensiva, memória ou armazenamento. Além disso, os recursos de elasticidade permitem aumentar ou reduzir a capacidade conforme a demanda, otimizando custos.

2.8 *Elastic Container Registry*

O Amazon *Elastic Container Registry* (ECR) é um serviço gerenciado da AWS que permite armazenar, compartilhar e implantar imagens de contêiner de forma segura e escalável. O ECR facilita a implantação de aplicações conteinerizadas sem a necessidade de gerenciar repositórios próprios. Ele oferece suporte à autenticação por meio do AWS *Identity and Access Management* (IAM), garantindo que apenas usuários autorizados possam acessar as imagens. Além disso, o ECR inclui recursos como varredura de vulnerabilidades e alta disponibilidade global, proporcionando uma solução confiável para gerenciar imagens de contêiner em qualquer estágio do ciclo de vida da aplicação (Rahman, 2023a).

2.8.1 Elastic Container Service

O Amazon *Elastic Container Service* (ECS) é um serviço gerenciado da AWS para executar e escalar contêineres Docker em ambientes altamente seguros e de alto desempenho. Com suporte integrado ao Docker, o ECS permite que as organizações implantem rapidamente aplicações conteinerizadas, eliminando a complexidade de configurar *clusters* manualmente. Ele se integra profundamente a outros serviços da AWS, como Amazon CloudWatch para monitoramento, IAM para controle de acesso e *Elastic Load Balancing* para distribuição de tráfego. Essa integração nativa, combinada com suporte para diferentes tipos de implantação, como *blue/green* e *rolling updates*, torna o ECS uma escolha confiável para ambientes produtivos e de desenvolvimento (Rahman, 2023a).

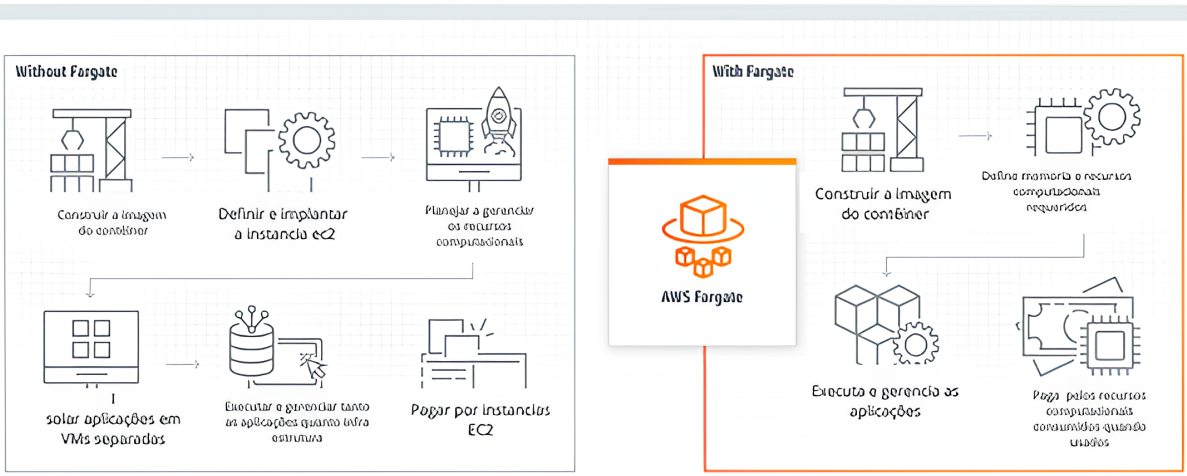
O ECS é conhecido por sua flexibilidade, permitindo que os usuários escolham entre diferentes modos de execução, como ECS com instâncias EC2 ou ECS Fargate. Com o modo baseado em EC2, os usuários podem configurar e otimizar manualmente o *hardware* subjacente, garantindo maior controle sobre a infraestrutura. Por outro lado, o modo Fargate é ideal para quem busca um ambiente de computação sem servidor, onde a AWS gerencia toda a infraestrutura automaticamente. Além disso, o ECS é altamente escalável, permitindo gerenciar

aplicações em larga escala enquanto reduz a sobrecarga operacional. A facilidade de integração com redes privadas e públicas, por meio da Amazon VPC, também reforça sua capacidade de atender a diversos casos de uso (Rahman, 2023a).

2.8.1.1 Fargate

O Amazon ECS Fargate (Rahman, 2023a) é uma solução para computação sem servidor da AWS que simplifica o gerenciamento de contêineres, eliminando a necessidade de configurar ou gerenciar servidores ou *clusters*. Com o Fargate, os usuários podem concentrar-se apenas na execução de suas aplicações, já que o serviço cuida do provisionamento, escalonamento e manutenção da infraestrutura subjacente. Essa abordagem oferece escalabilidade automática, cobrando apenas pelos recursos consumidos durante a execução dos contêineres, o que torna o Fargate ideal para aplicações modernas que demandam alta agilidade e flexibilidade.

Figura 3 – Arquitetura *Fargate*



Fonte: Amazon Web Services (2025c)

A Figura 3 apresenta uma comparação entre o uso de AWS Fargate e a execução tradicional de contêineres em instâncias EC2 para a implantação de aplicações baseadas em contêineres. A figura divide o processo em duas abordagens.

Sem o Fargate, o processo envolve criar uma imagem de contêiner e, em seguida, definir e implantar instâncias EC2 para executar a aplicação. Os recursos computacionais e de memória precisam ser provisionados e gerenciados manualmente. Cada aplicação é isolada em máquinas virtuais separadas (VMs). Além da aplicação, é necessário gerenciar a infraestrutura da instância EC2. O custo é baseado no uso das instâncias EC2, independentemente da real

necessidade dos recursos.

Com o Fargate o processo ainda exige a criação da imagem do contêiner, mas elimina a necessidade de gerenciar servidores EC2 em vez de provisionar instâncias EC2, o usuário apenas define a quantidade necessária de memória e CPU. A execução da aplicação ocorre sem a necessidade de gerenciar a infraestrutura subjacente. O custo é baseado nos recursos efetivamente utilizados, garantindo maior eficiência econômica.

A imagem 3 destaca que AWS Fargate simplifica o gerenciamento de contêineres, eliminando a necessidade de gerenciar servidores EC2 e permitindo que os desenvolvedores foquem apenas na execução das aplicações. Com Fargate, a cobrança é feita com base no consumo real de recursos, oferecendo maior flexibilidade e escalabilidade para aplicações baseadas em contêineres.

2.8.1.2 Instância EC2

O Amazon ECS (Tihfon *et al.*, 2016) também permite o gerenciamento de contêineres em instâncias EC2, oferecendo maior controle sobre o ambiente subjacente. Nesse modelo, os usuários podem configurar manualmente o tipo de instância, tamanho, e número de nós para o *cluster*, além de otimizar custos utilizando instâncias reservadas ou de baixo custo, como as *Spot Instances*. Embora exija mais esforço operacional em comparação com o Fargate, o modelo baseado em EC2 é vantajoso para cenários em que é necessário personalizar o ambiente ou aproveitar capacidades específicas das instâncias.

2.9 Computação sem servidor

A computação sem servidor, representa uma mudança significativa no paradigma de desenvolvimento e implementação de aplicações em ambientes de computação em nuvem. Segundo Moreira *et al.* (2021) a computação sem servidor é uma abordagem na qual os clientes não precisam administrar os servidores nos quais seus aplicativos são executados.

Este modelo permite que desenvolvedores escrevam e implantem código sem se preocupar com a infraestrutura subjacente, como servidores ou máquinas virtuais, que tradicionalmente precisariam ser configuradas e gerenciadas. Ao invés disso, a alocação e gerenciamento dos recursos de *hardware* são abstraídos pelo provedor de serviços de nuvem, o que proporciona uma série de vantagens e desafios para o desenvolvimento de software.

O termo "sem servidor"pode parecer de certa forma ambíguo, pois ainda existem servidores físicos e virtuais envolvidos na execução das aplicações. A principal diferença é que, no modelo de computação sem servidor, o gerenciamento desses servidores é feito inteiramente pelo provedor de nuvem, como Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure ou outros. O desenvolvedor se preocupa apenas com o código e as funções que precisam ser executadas, enquanto o provedor cuida automaticamente da escalabilidade, da disponibilidade e da manutenção da infraestrutura.

De acordo com Amazon Web Services (2024), a adoção da tecnologia de computação sem servidor traz diversas vantagens, incluindo a adaptação em escala, que permite escalabilidade automática de zero até demandas de pico, tornando-a especialmente eficaz para aplicações de Aprendizado de máquina, onde o fluxo de dados pode variar significativamente. Além disso, há uma redução significativa de custos, já que o pagamento é feito apenas pelo uso efetivo dos recursos. Outra vantagem é a facilidade de desenvolvimento, pois as aplicações sem servidor vêm com integrações nativas que eliminam a necessidade de configurar a infraestrutura, permitindo que os desenvolvedores se concentrem exclusivamente no código e na lógica de negócio.

2.10 *Offloading* Computacional

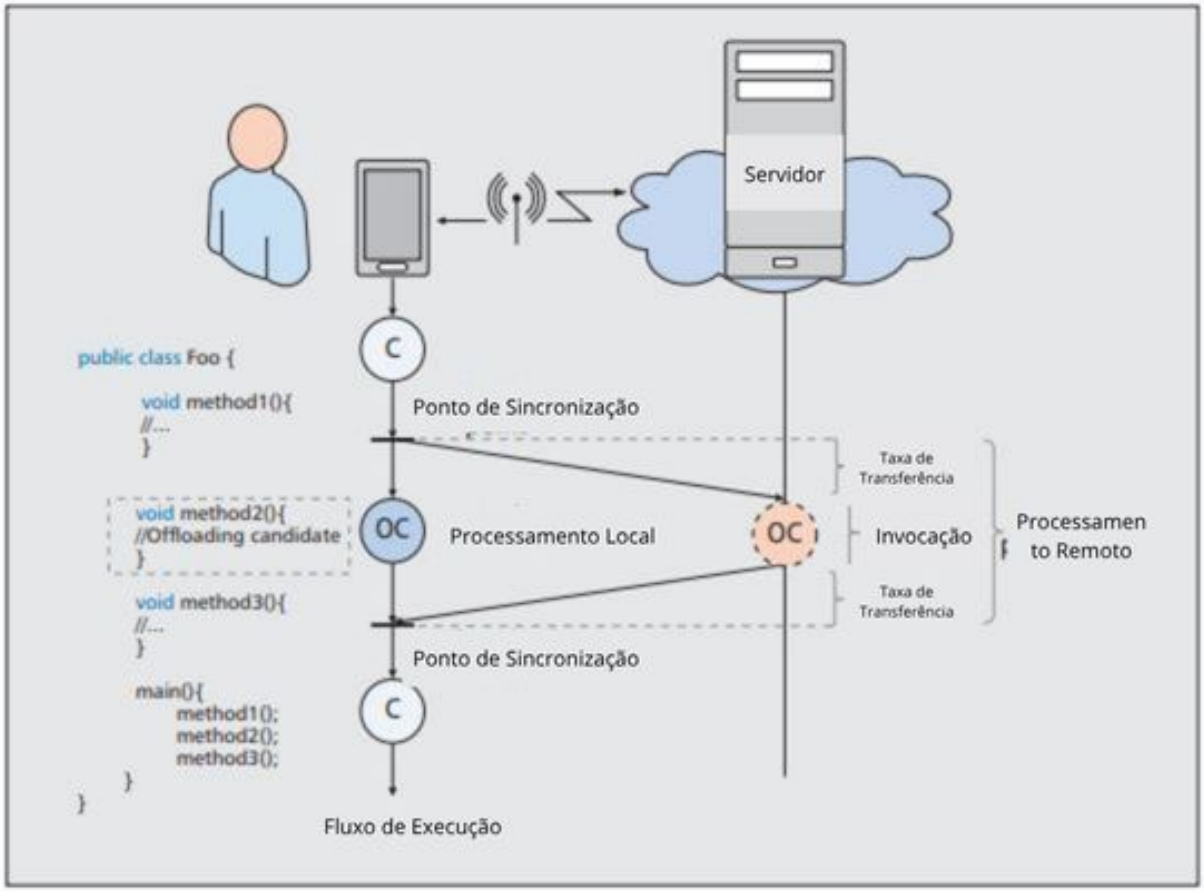
O conceito de *offloading* computacional surge como uma solução inovadora para superar as limitações de recursos computacionais em dispositivos móveis, particularmente em termos de processamento e consumo de energia. De acordo com Flores *et al.* (2015), *offloading* computacional refere-se ao processo oportunista em que um dispositivo móvel delega a execução de determinadas porções de código a servidores remotos, otimizando a utilização de recursos e preservando a autonomia da bateria do dispositivo.

No cenário moderno de computação móvel, os dispositivos como *smartphones* e *tablets* são frequentemente limitados por seu poder computacional e capacidade de bateria. Isso se torna particularmente evidente em aplicações que exigem processamento intensivo, como jogos, reconhecimento de voz, e algoritmos de Aprendizado de Máquina. O *offloading* computacional propõe uma abordagem onde essas tarefas intensivas são detectadas automaticamente pelo dispositivo e, quando necessário, delegadas para execução em um servidor remoto mais poderoso, que pode processar as tarefas com maior eficiência energética e computacional.

A arquitetura de *offloading* computacional, conforme ilustrada na Figura 4 exemplifica um modelo de computação em nuvem onde a carga de processamento é transferida do

dispositivo cliente para um servidor remoto. Esse modelo é projetado para manter a interatividade do aplicativo e a funcionalidade geral através de sincronizações eficazes e um gerenciamento inteligente da transferência de dados. A lógica de decisão, embutida no dispositivo, é responsável por monitorar continuamente o consumo de recursos e determinar a viabilidade do *offloading* computacional.

Figura 4 – Arquitetura *Offloading* Computacional



Fonte: Flores *et al.* (2015)

Como mostrado juntamente com os resultados do artigo do autor Barbosa *et al.* (2021) Os benefícios do *offloading* computacional são múltiplos:

- **Economia de Energia**: Ao delegar tarefas intensivas a servidores remotos, o dispositivo móvel reduz o esforço computacional local, resultando em uma economia significativa de energia e prolongando a vida útil da bateria.
- **Melhor Desempenho**: Servidores remotos geralmente possuem recursos de *hardware* mais avançados, capazes de processar tarefas complexas de maneira mais rápida e eficiente do que os dispositivos móveis.

- **Flexibilidade e Escalabilidade**: O *offloading* computacional permite que dispositivos móveis executem aplicativos mais complexos, que seriam inviáveis de outra forma, proporcionando uma escalabilidade que se adapta à demanda computacional.

Apesar das vantagens, o *offloading* computacional também apresenta desafios. A dependência da conectividade de rede é um fator crítico, já que a transferência de dados entre o dispositivo e o servidor remoto requer uma conexão estável e rápida. Além disso, a latência introduzida pela comunicação de rede pode impactar negativamente a experiência do usuário se não for adequadamente gerenciada. De acordo com Flores *et al.* (2015) alguns deles são:

- **Sustentabilidade Prática:** "A sustentabilidade do *offloading* computacional em prática é uma questão em aberto."(Flores *et al.*, 2015).
- **Esforço Computacional**: "Computacional *offloading* em ambientes reais pode impor mais esforço computacional no móvel em vez de reduzir a carga de processamento."(Flores *et al.*, 2015).
- **Políticas de Alocação de Recursos**:"A alocação de recursos não pode ocorrer indiscriminadamente, pois o uso de recursos computacionais é associado a um custo"(Flores *et al.*, 2015).
- **Monitoramento e Limites de Invocação**: "Um pedido de *offloading* computacional de código não pode ser tratado de forma indiferente. A invocação remota de um método deve ser monitorada sob diferentes *throughput* dos sistemas"(Flores *et al.*, 2015).

2.11 Métricas de Avaliação

Para realizar uma análise comparativa robusta entre as plataformas AWS Fargate e EC2, foram selecionadas três métricas chave que, em conjunto, oferecem uma visão holística do desempenho, da experiência do usuário e da viabilidade econômica da solução de *offloading*. A escolha destas métricas é fundamentada em práticas consolidadas na avaliação de desempenho de sistemas em nuvem (Rahman, 2023b; Jain, 1991).

- **Latência Individual (Tempo Médio de Inferência por Imagem):** Esta métrica mede o tempo de resposta de ponta a ponta para uma única requisição, desde o envio da imagem pelo *frontend* até o recebimento do resultado da classificação. A latência é um indicador direto da responsividade do sistema e impacta fundamentalmente a experiência do usuário final (UX), sendo crucial para aplicações interativas em tempo real (Buyya *et al.*, 2009).
- **Tempo Total de Processamento:** Corresponde ao tempo total necessário para processar

sequencialmente todo o conjunto de 600 imagens. Esta métrica avalia a vazão (*throughput*) do sistema sob uma carga de trabalho contínua, indicando a capacidade e a eficiência de cada plataforma para lidar com o processamento em lote. Um tempo total menor significa uma maior capacidade de processamento no mesmo intervalo.

– **Custo Financeiro por Execução:** Refere-se ao custo monetário direto, em dólares (USD), associado aos recursos AWS consumidos durante a execução de um teste completo. Esta métrica é essencial para avaliar a viabilidade econômica e a eficiência de custo de cada abordagem, um fator decisivo para a implantação de soluções em ambientes de produção. A análise de custo é particularmente relevante ao comparar o modelo de pagamento por uso (*pay-as-you-go*) do Fargate com o modelo de provisionamento do EC2 (Eyk *et al.*, 2018).

A análise conjunta dessas três métricas permite uma avaliação completa do *trade-off* entre desempenho (latência e vazão) e custo, fornecendo os *insights* necessários para determinar a arquitetura mais adequada para o cenário proposto.

3 TRABALHOS RELACIONADOS

Na seção de trabalhos relacionados, exploramos diversas pesquisas que estabelecem a fundação teórica e prática para a aplicação de técnicas de *offloading* computacional e computação sem servidor em contextos de Aprendizado de Máquinas. Este segmento do trabalho é crucial para entender como desenvolvimentos recentes e estudos anteriores influenciam e suportam a proposta atual de permitir que dispositivos de baixo poder computacional efetivamente processem tarefas computacionalmente intensivas. Analisaremos as abordagens previamente documentadas, comparando-as com as metodologias propostas neste trabalho, destacando tanto os avanços quanto os desafios remanescentes que moldam nossa pesquisa.

3.1 Acelerando a identificação de impressões digitais através da técnica do *offloading* computacional

O estudo em questão abordado pelo autor Barbosa e Rego (2019) fala sobre utilização de *offloading* computacional para melhorar a eficiência de sistemas de reconhecimento de impressões digitais em dispositivos com recursos limitados, como *smartphones* e fechaduras inteligentes. O reconhecimento de impressões digitais é uma tecnologia crítica para a segurança, mas enfrenta desafios quando os dispositivos têm capacidades computacionais e de armazenamento restritas(Barbosa; Rego, 2019).

O estudo apresenta um caso detalhado sobre a implementação de *offloading* computacional para acelerar a identificação de pessoas por meio de leitores de impressões digitais integrados a *smartphones*. Utiliza-se a arquitetura que combina o uso de nuvem e *cloudlets* para processar dados de impressões digitais, demonstrando como a carga computacional pode ser transferida de dispositivos locais para sistemas mais robustos na nuvem.

Comparativamente, o presente estudo explora uma abordagem semelhante ao para otimizar aplicações de aprendizado de máquinas em dispositivos de baixo poder. No entanto, enquanto o estudo referenciado foca especificamente no reconhecimento de impressões digitais, a seguinte pesquisa visa aplicar técnicas de *offloading* computacional em um espectro mais amplo de aplicações de ML, abrangendo não apenas segurança, mas também outras funcionalidades dependentes de processamento intensivo. Os resultados promissores do estudo citado, que indicam uma redução de tempo em até 85%, reforçam a viabilidade do *offloading* computacional como uma estratégia eficaz para melhorar o desempenho de dispositivos limitados, um princípio

que também é central no meu projeto.

3.2 Avaliação das Linguagens Python e Go na Plataforma AWS Lambda para Microsserviços de Computação de Alto Desempenho

O estudo de Moreira *et al.* (2021) investiga a influência da escolha da linguagem de programação no desempenho de aplicações desenvolvidas em ambientes de computação sem servidor, especificamente na plataforma Amazon Web Services (AWS). O estudo se concentra nas linguagens Python e Go, destacando-se por abordar como essas linguagens podem afetar a execução de aplicações de alto desempenho, especialmente em tarefas de álgebra linear.

A pesquisa realiza experimentos práticos para comparar o desempenho de Python, usando a biblioteca NumPy, com Go em um contexto sem servidor, visando identificar qual das linguagens oferece melhor eficiência operacional. A escolha de Python e Go é significativa, dado que ambas as linguagens possuem características distintas em termos de gerenciamento de memória, concorrência e facilidade de uso, fatores cruciais em ambientes de computação sem servidor(Moreira *et al.*, 2021).

Com base nos resultados apresentados no trabalho, a avaliação de desempenho entre as linguagens Python e Go na plataforma AWS Lambda para multiplicação de matrizes revelou conclusões distintas dependendo do uso de bibliotecas. Na comparação entre as linguagens puras (sem bibliotecas otimizadas), Go demonstrou uma performance muito superior, conseguindo processar matrizes com ordens (dimensões) em média três vezes maiores que as de Python sob as mesmas restrições de tempo e memória. Contudo, ao utilizar bibliotecas numéricas especializadas, o cenário se inverteu: Python, com o uso da biblioteca NumPy, alcançou o melhor desempenho geral, superando tanto o Go puro quanto o Go com a biblioteca GoNum. O estudo conclui que, para aplicações de alto desempenho em um ambiente computação sem servidor como o AWS Lambda, a combinação de Python com NumPy é a alternativa mais eficiente, mas se a escolha for entre as linguagens nativas, Go é a opção vencedora.

Em comparação com este trabalho, que também explora a computação sem servidor, mas com foco em técnicas de *offloading* computacional para otimizar aplicações de aprendizado de máquina, o estudo em questão proporciona uma base comparativa valiosa sobre como a escolha da linguagem de programação pode impactar o desempenho de sistemas sem servidor. Embora ambos os trabalhos compartilhem o objetivo de maximizar o desempenho nas nuvens, a análise específica das linguagens em diferentes cenários de aplicação, como feito no estudo

referenciado, complementa e enriquece a abordagem da presente pesquisa. Este contraste destaca a importância de escolher adequadamente a tecnologia base, tanto em termos de linguagem de programação quanto de estratégias de *offloading* computacional, para alcançar a eficiência desejada em ambientes computacionais restritos.

3.3 Análise Comparativa de Desempenho e Custo em Serviços de Contêineres Sem Servidor

Em um estudo abrangente e recente, Rahman (2023a) realiza uma análise comparativa de desempenho, custo e experiência do desenvolvedor em serviços de contêineres em nuvem. O trabalho foca em três plataformas de computação sem servidor: Google Cloud Platform (GCP) Cloud Run (PaaS), Google Kubernetes Engine (GKE) AutoPilot (CaaS) e AWS EKS com AWS Fargate (CaaS), que é diretamente relevante para a presente pesquisa.

A metodologia de Rahman envolveu o desenvolvimento e a implantação de uma aplicação web nessas três plataformas para avaliar suas diferenças. Foram conduzidos testes de carga incrementais e de pico para analisar o desempenho, enquanto os custos foram comparados com base na utilização de recursos durante os testes. A experiência do desenvolvedor foi avaliada com base na complexidade de implementação e uso dos serviços.

Os principais resultados do estudo indicam que, no desempenho, a solução baseada em PaaS (GCP Cloud Run) apresentou, em média, menor latência e maior taxa de transferência (*throughput*) em comparação com as soluções CaaS. No entanto, o AWS EKS com Fargate demonstrou um desempenho sólido, com latências de inicialização a frio (cold start) mais baixas em comparação com o GKE AutoPilot. No custo, o GCP Cloud Run se mostrou a opção mais econômica entre as três plataformas avaliadas nos cenários de teste, em grande parte devido à sua menor latência e maior eficiência. Na experiência do desenvolvedor, Plataformas PaaS, como o Cloud Run, oferecem uma experiência de desenvolvimento mais simples, exigindo pouco ou nenhum conhecimento de tecnologias subjacentes. Em contrapartida, as plataformas CaaS (Fargate e GKE AutoPilot) demandam um conhecimento aprofundado de Kubernetes para otimizar o uso de recursos, segurança e desempenho.

Em comparação com o presente trabalho, o estudo de Rahman (2023) oferece um valioso panorama sobre o desempenho do AWS Fargate em relação a outras plataformas de contêineres sem servidor. Enquanto Rahman compara diferentes modelos de serviço (PaaS vs. CaaS), esta pesquisa foca em uma comparação distinta: a avaliação do AWS Fargate (uma

abordagem de contêineres sem servidor) contra a execução de contêineres em instâncias EC2 (uma abordagem IaaS mais tradicional e gerenciada pelo usuário). Além disso, o foco deste trabalho é mais específico, aplicando essas tecnologias ao problema de *offloading* computacional para aplicações de aprendizado de máquina em dispositivos de baixo poder, um caso de uso que não é o foco principal do estudo de Rahman.

3.4 *Modernizing Applications with Containers in Public Cloud*

No estudo apresentado por Chen (2019), o artigo "Modernizing Applications with Containers in Public Cloud" explora como os contêineres revolucionaram a arquitetura de aplicações e métodos de desenvolvimento, trazendo agilidade e escalabilidade para empresas em sua transição para a computação em nuvem. O trabalho destaca que os contêineres, como tecnologia de virtualização leve, permitem isolar aplicações de maneira eficiente, facilitando a portabilidade e a consistência em diferentes ambientes. A integração com tecnologias modernas, como DevOps, microsserviços e CI/CD (Integração Contínua/Entrega Contínua), potencializa a capacidade de desenvolvimento rápido e confiável de software, características essenciais para empresas que buscam inovação contínua.

Chen (2019) enfatiza a importância do uso de contêineres em ambientes de nuvem pública, que oferecem infraestrutura escalável, armazenamento persistente e redes avançadas. O artigo também discute como os contêineres ajudam a modernizar aplicações legadas por meio de *replatforming* e refatoração, garantindo maior eficiência e adaptabilidade. Com um crescimento anual de 79% projetado até 2021 para instâncias de contêineres, o estudo apresenta cenários que vão desde a adoção de práticas de microsserviços até a utilização de novas abordagens como computação sem servidor. Além disso, Chen (2019) detalha as vantagens do uso de orquestradores, como Kubernetes, para automatizar processos e garantir alta disponibilidade, escalabilidade e eficiência operacional.

3.5 Análise Comparativa

A análise dos trabalhos relacionados (Barbosa; Rego, 2019; Moreira *et al.*, 2021; Chen, 2019; Rahman, 2023a) revela uma base para a compreensão do *offloading* computacional e da computação sem servidor em nuvem. Para contextualizar a contribuição do presente estudo, os Quadros 1, 2 e 3 apresentam um comparativo detalhado das principais características desses

trabalhos.

Quadro 1 – Comparativo de Aspectos Conceptuais e de Aplicação dos Trabalhos Relacionados

Trabalho	**Tecnologia Central**	**Foco da Aplicação**	**Problema Principal**
Barbosa e Rego (2019)	Offloading Computacional	Reconhecimento Digital	Eficiência em disp. limitados
Moreira *et al.* (2021)	Computação Sem Servidor	Microsserviços/Alto Desempenho	Desempenho de Linguagens
Chen (2019)	Contêineres	Modernização de Aplicações	Agilidade/Escalabilidade de SW
Rahman (2023a)	Computação Sem Servidor	Aplicações Web Genéricas	Desempenho/Custo/Exp. Desenvolvedor
Presente Trabalho	**Offloading Computacional**	**Classificação de Imagem (ML)**	**Limitações de Disp. com Baixo Poder**

Fonte: Elaborado pelo Autor

Quadro 2 – Comparativo de Aspectos de Implementação e Infraestrutura dos Trabalhos Relacionados

Trabalho	**Computação Sem Servidor**	**Contêineres**	**Provedor de Nuvem**
Barbosa e Rego (2019)	Não (Geral)	Implícito/Não Foco	Não Especificado
Moreira *et al.* (2021)	Sim (AWS Lambda - FaaS)	Implícito/Não Foco	AWS
Chen (2019)	Conceitual (Tendência)	Sim (Docker, Kubernetes)	Nuvem Pública (AWS, Azure)
Rahman (2023a)	Sim (Cloud Run - PaaS, Fargate)	Sim (Docker, Kubernetes)	AWS, GCP
Presente Trabalho	**Sim (AWS Fargate - CaaS)**	**Sim (Docker, ECS, Fargate/EC2)**	**AWS**

Fonte: Elaborado pelo Autor

Como observado nos Quadros 1 e 2, a computação sem servidor e o uso de contêineres são temas recorrentes na literatura recente. O estudo de Rahman (2023a) se alinha diretamente a este trabalho na avaliação de plataformas de computação sem servidor baseadas em contêineres, como o AWS Fargate. No entanto, enquanto Rahman foca em aplicações web genéricas, o presente estudo aprofunda-se na aplicação dessas tecnologias para a classificação de imagem utilizando Aprendizado de Máquina, um caso de uso mais específico e com implicações diretas para dispositivos de baixo poder.

Quadro 3 – Comparativo de Trabalhos por Abordagem Metodológica e Métricas

Trabalho	**Abordagem Metodológica**	**Métricas Chave**
Barbosa e Rego (2019)	Implementação Prática	Tempo de Processamento
Moreira *et al.* (2021)	Experimento Comparativo	Desempenho Runtime
Chen (2019)	Análise Conceitual/Tendências	Agilidade, Escalabilidade
Rahman (2023a)	Experimento Comparativo (CaaS vs PaaS)	Latência, Throughput, Custo, Exp. Desenvolvedor
Presente Trabalho	**Experimento Comparativo (Fargate vs EC2)**	**Tempo Total, Tempo Médio, Custo Financeiro**

Fonte: Elaborado pelo Autor

o Quadro 3 ilustra as distintas metodologias e métricas adotadas. Enquanto alguns trabalhos empregam análises conceituais (Chen, 2019), a maioria dos estudos mais recentes, incluindo este TCC, opta por experimentos comparativos em ambientes de nuvem.

A contribuição da presente pesquisa se manifesta na comparação prática e direta do modelo de computação sem servidor (AWS Fargate) com uma abordagem mais tradicional baseada em IaaS (AWS EC2) para a execução de contêineres. Esta análise é especificamente aplicada ao contexto do *offloading* computacional de aplicações de Aprendizado de Máquina,

fornecendo métricas de desempenho e, crucialmente, de custo financeiro que são essenciais para a tomada de decisões em cenários de recursos computacionais limitados.

4 METODOLOGIA

Esta seção detalha os procedimentos e as ferramentas empregadas para investigar a eficácia das tecnologias de computação sem servidor e de *offloading* computacional na otimização da classificação de imagem em dispositivos de baixa capacidade computacional. A metodologia abrange desde a definição da arquitetura e desenvolvimento da aplicação protótipo até a execução dos experimentos e a coleta de métricas, visando validar as hipóteses e as soluções propostas.

O fluxo metodológico adotado pode ser visualizado no Fluxograma da Figura5.

Figura 5 – Fluxograma Metodologia.

Fonte: Elaborado pelo Autor

4.1 Definição de Volume de imagens e métricas de avaliação

Volume de Imagens

Para cada teste, utiliza-se um conjunto de imagens com as seguintes características:

– **Quantidade total de imagens:** Um conjunto de dados com 600 imagens de animais

terrestres e marinhos

- **Tamanhos de arquivo:** Imagens com tamanhos variando entre **2KB a 5MB**, simulando diferentes resoluções e formatos encontrados em aplicações reais.
- **Lotes de processamento:** As imagens foram enviadas individualmente, ou seja, uma imagem por requisição, reproduzindo o comportamento típico de sistemas de inferência em tempo real.

Métricas Coletadas

As seguintes métricas de desempenho e custo são coletadas ao longo dos testes:

- **Latência Individual (Tempo Médio de Inferência por Imagem):** Tempo decorrido desde o envio da imagem pelo *frontend* até o recebimento do resultado da inferência pela API. Esta métrica é essencial para avaliar a experiência do usuário final.
- **Tempo Total de Processamento:** Tempo total necessário para que um conjunto de imagens seja processado de forma sequencial.
- **Custo Financeiro por Execução:** Custo direto associado à execução dos contêineres e aos recursos AWS utilizados para processar o conjunto de imagens. As informações são obtidas a partir do *AWS Billing* e complementadas com análise de uso via métricas da nuvem.

4.2 Definição dos Serviços AWS

Para a implementação e avaliação da solução de *offloading* computacional, selecionam-se e configuram-se selecionadas e configuradas duas abordagens distintas para a execução de contêineres na plataforma Amazon Web Services (AWS): o AWS Elastic Container Service (ECS) com Fargate e o AWS ECS com instâncias EC2.

- **AWS ECS com Fargate (Abordagem de Computação Sem Servidor):** Esta modalidade representa uma solução de computação sem servidor, onde o AWS Fargate gerencia automaticamente a infraestrutura subjacente (servidores, *clusters* e escalabilidade). O código da aplicação é executado sob demanda, e o pagamento é feito apenas pelos recursos de CPU e memória efetivamente consumidos. Essa abordagem simplifica o gerenciamento da infraestrutura e oferece escalabilidade elástica.
- **AWS ECS com Instâncias EC2 (Abordagem Baseada em IaaS Gerenciada):** Esta

abordagem permite a execução de contêineres Docker em instâncias do Amazon EC2 gerenciadas pelo ECS. Diferente do Fargate, o usuário possui maior controle e responsabilidade sobre o provisionamento, configuração e escalabilidade das instâncias EC2 subjacentes. Esta opção é crucial para a comparação, pois serve como um referencial que ilustra o *trade-off* entre o gerenciamento manual/custo fixo (EC2) e a abstração/*pay-as-you-go* (Fargate).

A escolha dessas duas plataformas visa comparar diretamente os benefícios de escalabilidade automática e simplicidade operacional do AWS Fargate com a flexibilidade e o maior controle de recursos oferecidos pelas instâncias EC2, no contexto de um serviço de inferência de aprendizado de máquina. A infraestrutura básica para ambas as abordagens (*VPC, subnets, Internet Gateway, Route Table e Security Group*) foi provisionada utilizando Terraform, conforme detalhado na Seção 4.3.

4.3 Desenvolvimento de um protótipo de Aplicação com Modelos Pré-Treinados

Para simular o processo de *offloading* computacional, foi desenvolvida uma aplicação protótipo composta por uma API RESTful e uma interface de usuário web simples. A API é responsável por receber imagens, processá-las utilizando um modelo de aprendizado de máquina pré-treinado e retornar os resultados da classificação.

API de Inferência (Backend)

- **Tecnologia:** A API é desenvolvida em **Python** utilizando o *microframework* **Flask**, devido à sua leveza e flexibilidade para construção de serviços web e APIs.
- **Modelo de Aprendizado de Máquina:** Para a tarefa de classificação de imagem, emprega-se o modelo **MobileNetV3Large**, com pesos pré-treinados no conjunto de dados **ImageNet**. O *MobileNetV3Large* é um modelo de rede neural convolucional (CNN) conhecido por sua eficiência computacional e tamanho reduzido, tornando-o adequado para cenários de inferência rápida e para dispositivos de baixa capacidade. Embora o *MobileNetV2* seja uma opção, o *MobileNetV3Large* representa uma evolução com melhor balanceamento entre latência e precisão, o que o torna mais pertinente para testes de desempenho em um ambiente de *offloading* computacional.
- **Bibliotecas:** Utilizou-se **TensorFlow** e **Keras** para o carregamento e execução do modelo,

e **Numpy** para manipulação de *arrays*.

- **Funcionalidades:** A API expõe dois *endpoints*:
  - `GET /`: Um *health check* simples para verificar se o serviço está ativo e o modelo carregado.
  - `POST /analyze`: Recebe uma imagem via `multipart/form-data`, pré-processa-a (redimensionando para 224x224 pixels e aplicando a normalização específica do MobileNetV3), realiza a inferência do modelo e retorna as três principais classes previstas com suas respectivas probabilidades em formato `JSON`.
- **Carregamento do Modelo:** O modelo *MobileNetV3Large* é carregado uma única vez na inicialização da aplicação (e do contêiner), garantindo que as inferências subsequentes sejam mais rápidas, sem a sobrecarga de carregamento do modelo a cada requisição.
- **Contenerização:** A aplicação Flask foi empacotada em uma imagem de contêiner Docker (`Dockerfile`), garantindo a portabilidade e consistência do ambiente de execução entre as plataformas **AWS Fargate** e **EC2**.

Interface de Usuário (Frontend - Execução Local)

- **Tecnologia:** Desenvolveu-se uma interface web minimalista em **HTML**, **CSS** e **JavaScript** puro.
- **Funcionalidade:** Permite ao usuário selecionar uma imagem local e enviá-la para o *endpoint* `/analyze` da API implantada na nuvem. Os resultados da classificação (classes e probabilidades) são exibidos na própria página.
- **Propósito:** Este *frontend* serve como o *"dispositivo de baixa capacidade computacional"* simulado, onde o processamento intensivo da imagem (a inferência de ML) é descarregado para o serviço de nuvem, demonstrando o conceito de *offloading* computacional. A execução do *frontend* localmente isola a medição do desempenho da inferência na nuvem.

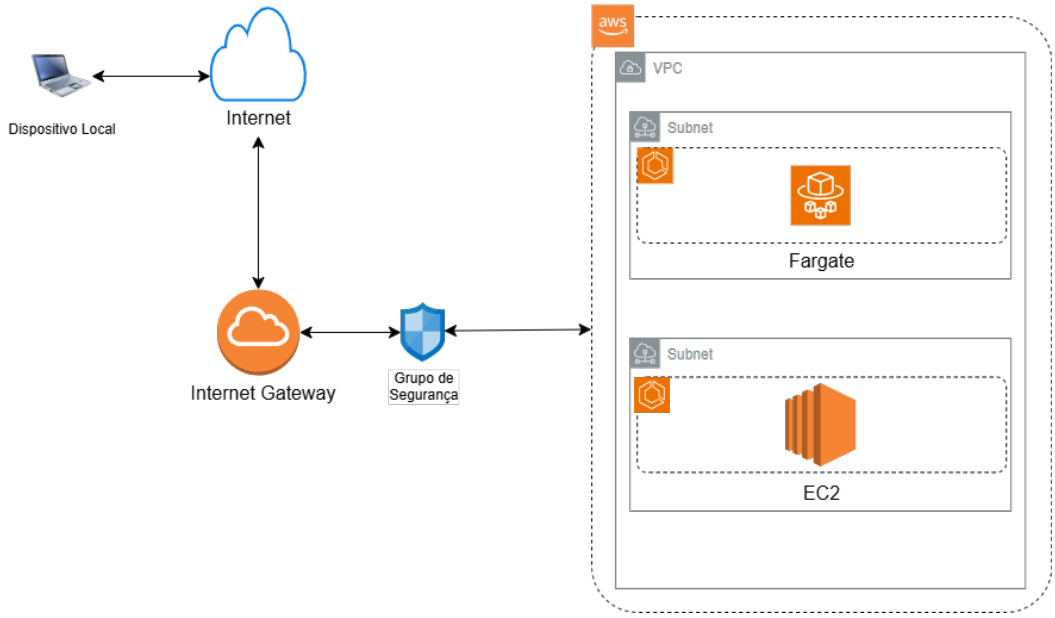
Todo o código-fonte da API de inferência, da interface web do cliente e dos arquivos Terraform para o provisionamento da infraestrutura AWS encontra-se disponível no repositório do projeto em: offloading-ml-containers-aws

4.4 Arquitetura de Rede para *Offloading* Computacional

Para viabilizar o *offloading* computacional, projetou-se e provisionou-se uma arquitetura de rede dedicada na **AWS**, utilizando o **Terraform** como ferramenta de *Infrastructure as Code* (IaC). Esta abordagem garante a reprodutibilidade, versionamento e automação do ambiente.

O diagrama de componentes da arquitetura geral pode ser visualizado na **Figura 6**.

Figura 6 – Diagrama de Arquitetura.



Fonte: Elaborado pelo Autor

Componentes da Infraestrutura de Rede

- ***Virtual Private Cloud* (VPC):** Uma rede virtual isolada (`10.0.0.0/16`) para hospedar todos os recursos da aplicação, garantindo segurança e segmentação.
- ***Subnet Pública*:** Uma sub-rede (`10.0.1.0/24`) configurada para permitir o acesso público aos serviços da API.
- ***Internet Gateway* (IGW):** Permite a comunicação entre a VPC e a Internet.
- **Tabela de Rotas:** Define o caminho do tráfego de rede, direcionando o tráfego de saída para o *Internet Gateway*.
- **(*Security Group*):** Atua como um *firewall* virtual, controlando o tráfego de entrada e saída.Configurou-se para permitir o tráfego `TCP` na porta `8080` (porta da aplicação) de qualquer origem (`0.0.0.0/0`), e todo o tráfego de saída.

- ***Elastic Container Registry* (ECR):** Repositório privado gerenciado para armazenar as imagens de contêiner Docker da aplicação API. A imagem `apiv3:latest` foi enviada para este repositório.

Provisionamento com Terraform

O Terraform é utilizado para descrever e provisionar os recursos de rede e o ambiente ECS. O código descreve a criação da *VPC, Subnets, , Route Tables, Security Groups* e os componentes ECS (*cluster*, definição de tarefa e serviço).

Os arquivos de *Infrastructure as Code* (IaC) em Terraform, que detalham o provisionamento da infraestrutura AWS, podem ser consultados em: Github.

Execução com AWS ECS Fargate

- A tarefa ECS (`app_task`) foi configurada com `launch_type = "FARGATE"`, especificando `CPU = "512"` (equivalente a 0.5 vCPU) e `Memory = "1024"` (1GB).
- O serviço ECS (`app_service`) garante que um número `desired_count = 1` de tarefas Fargate seja executado e acessível publicamente via IP atribuído.
- O mapeamento de portas (`containerPort = 8080`, `hostPort = 8080`) expõe a API.

Execução com AWS ECS Instâncias EC2

- Para os testes com EC2, o processo envolveu uma modificação no Terraform para provisionar um *cluster* ECS com instâncias EC2, em vez de Fargate.
- O *cluster* Fargate foi destruído e, em seu lugar, foi criado um novo *cluster* com tarefas em EC2, utilizando a mesma imagem Docker e configuração de rede (*VPC, Subnets, Security Groups*).
- As especificações das instâncias EC2 foram escolhidas para serem comparáveis em termos de vCPU e memória às alocações do Fargate (t3.medium)(Amazon Web Services, 2025a).

Resumo da Arquitetura

Essa arquitetura permite que a aplicação **frontend** (executada localmente) envie requisições de análise de imagem para a **API** hospedada na nuvem de forma segura e escalável, delegando o processamento intensivo para a infraestrutura AWS.

4.5 Aplicação do *Offloading* Computacional

O **offloading** computacional é o pilar desta pesquisa, permitindo que a inferência do modelo de aprendizado de máquina **MobileNetV3Large**, computacionalmente intensiva, seja executada na nuvem em vez de localmente em um dispositivo de baixa capacidade. O processo se desdobra da seguinte forma:

1. **Captura da Imagem no Dispositivo Local:** O usuário interage com a interface *frontend* (executada localmente em um navegador web) e seleciona uma imagem para análise.
2. **Preparação para o Offloading:** A imagem selecionada é encapsulada em um objeto `FormData` pelo JavaScript do *frontend*.
3. **Delegação da Computação (Requisição HTTP):** Uma requisição `HTTP POST` é enviada do *frontend* para o endpoint `/analyze` da API implantada na nuvem (exemplo: `http://54.196.13.2:8080/analyze`). Esta requisição carrega a imagem como parte do corpo da mensagem.
4. **Processamento na Nuvem:**
   - A requisição é recebida pelo serviço ECS (Fargate ou EC2) na AWS.
   - A API em Python (`app.py`) extrai a imagem da requisição.
   - A imagem é pré-processada e então alimentada ao modelo **MobileNetV3Large** (já carregado em memória no contêiner).
   - O modelo realiza a inferência para classificar a imagem, gerando as previsões de classes e probabilidades.
5. **Retorno do Resultado:** Os resultados da inferência são serializados em formato `JSON` e enviados de volta como resposta `HTTP` para o *frontend* local.
6. **Exibição no Dispositivo Local:** O *frontend* local recebe a resposta e exibe as classes previstas e suas probabilidades para o usuário.

Este ciclo demonstra a delegação bem-sucedida de uma tarefa pesada. A complexidade do processamento do modelo de aprendizado de máquina é abstraída do dispositivo local, que atua apenas como um *"cliente leve"*, melhorando a experiência do usuário e contornando as limitações de hardware do dispositivo.

A escolha entre **Fargate** e **EC2** impacta diretamente a **latência de ponta a ponta** e o **custo operacional** desse processo de *offloading*.

4.6 Testes e Validação

Metodologia Experimental

Para validar a solução de *offloading* computacional e comparar o desempenho e custo das abordagens **AWS Fargate** e **EC2**, foram conduzidos experimentos sistemáticos.

Cenários de Teste

Realizam-se múltiplos testes para cada plataforma (Fargate e EC2), repetindo a operação de *offloading* de imagens em diferentes horários do dia com o objetivo de capturar possíveis variações no ambiente de nuvem.

Execução dos Testes

Os testes são executados a partir do *frontend web* em ambiente local, simulando um dispositivo de baixa capacidade computacional. A comunicação com a API na nuvem foi realizada diretamente via o endereço público do serviço ECS (seja Fargate ou EC2), como por exemplo:

```
http://54.196.13.2:8080/analyze
```

4.7 Comparação de Desempenho e Custo

Os resultados obtidos nos testes foram apresentados em formato de gráficos e tabelas, destacando as diferenças entre as duas abordagens. O objetivo é determinar qual solução é mais adequada para aplicações de classificação de imagem em larga escala, considerando fatores como tempo de resposta, custo por execução, escalabilidade e consumo de recursos. Essa análise forneceu *insights* sobre as vantagens e limitações do uso de computação sem servidor versus contêineres gerenciados na AWS para esse tipo de aplicação.

5 RESULTADOS

Esta seção apresenta e discute os resultados dos experimentos conduzidos para avaliar o desempenho e o custo financeiro da utilização das plataformas AWS Fargate e AWS EC2 na execução de aplicações de Aprendizado de Máquina para classificação de imagem com a técnica de offloading computacional. As métricas coletadas incluem o tempo total de processamento, o tempo médio de inferência por imagem e o custo operacional, visando determinar a viabilidade e a eficiência de cada abordagem para dispositivos de baixa capacidade computacional.

5.1 Apresentação dos Dados Experimentais

Os experimentos foram realizados por meio da execução de múltiplos testes em cada plataforma (AWS Fargate e AWS EC2). Para garantir a robustez da análise, foram coletados dados de três repetições para cada cenário. A Tabela 1 resume as métricas de desempenho e custo para cada plataforma.

Tabela 1 – Comparativo Detalhado de Desempenho e Custo entre AWS Fargate e EC2

Plataforma	**Tempo Total (s)**	**Tempo Médio (s)**	**Custo (USD)**
Fargate	1.830,86	3,05	0,07
Fargate	1.837,39	3,06	0,07
Fargate	1.790,19	2,98	0,07
EC2	1.724,15	2,87	0,09
EC2	1.748,85	2,91	0,09
EC2	1.730,65	2,88	0,09

Fonte: Elaborado pelo Autor

Para uma análise mais consolidada, calculam-se as médias e os desvios padrão para as principais métricas de cada plataforma, conforme apresentado na Tabela 2.

Tabela 2 – Comparativo Consolidado de Desempenho e Custo Médio entre AWS Fargate e EC2

Plataforma	**Média TT (s)**	**DP TT (s)**	**Média TM (s)**	**DP TM (s)**	**Média Custo (USD)**	**DP Custo**
Fargate	1.819,48	25,68	3,03	0,04	0,07	0,00
EC2	1.734,55	12,72	2,89	0,02	0,09	0,00

Fonte: Elaborado pelo Autor

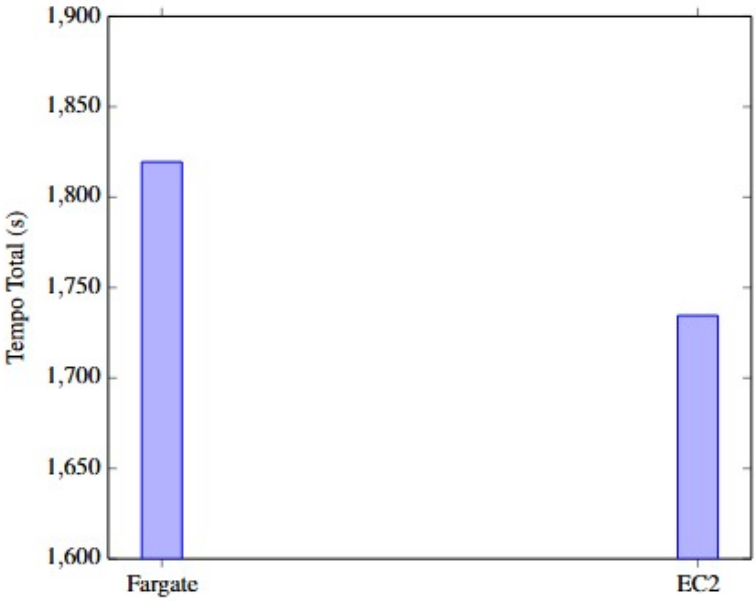
Nota: Os desvios padrão, embora calculados com base em uma amostra limitada de três repetições, indicam a variabilidade observada nas medições de cada plataforma.

5.2 Gráficos Comparativos

A seguir são apresentados os gráficos comparativos das principais métricas.

O Gráfico da Figura 7 apresenta uma comparação do tempo total médio necessário para processar o conjunto completo de 600 imagens em cada uma das plataformas avaliadas. Observa-se que a abordagem com instâncias EC2 obteve um desempenho ligeiramente superior, completando a tarefa em um tempo médio de 1734,55 segundos, enquanto a plataforma Fargate levou 1819,48 segundos. Essa diferença, embora marginal, sugere que o controle direto sobre a infraestrutura provisionada no modelo IaaS (EC2) pode oferecer uma pequena vantagem de desempenho para cargas de trabalho contínuas e previsíveis. A instância EC2, uma vez ativa, permanece "aquecida"e dedicada à tarefa, eliminando potenciais latências associadas ao provisionamento dinâmico de recursos que podem ocorrer no ambiente sem servidor do Fargate.

Figura 7 – Comparativo Tempo Total Médio.

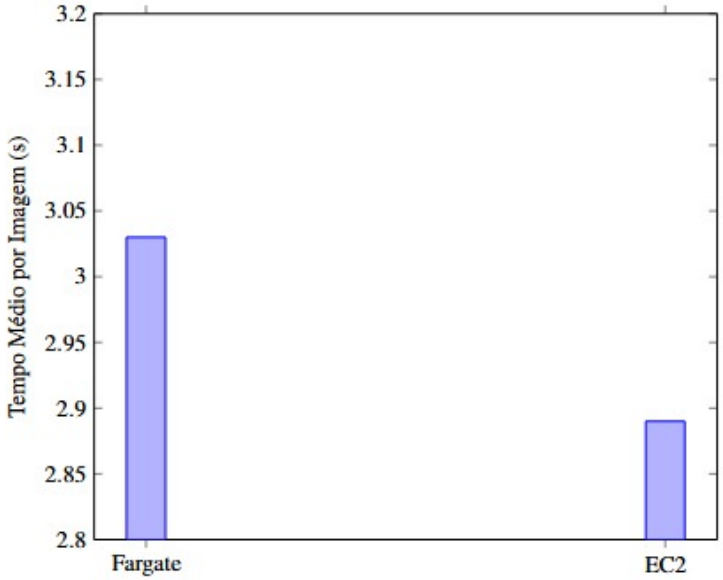


Fonte: Elaborado pelo Autor

O Gráfico da Figura 8 revela o aspecto financeiro da comparação, apresentando uma clara inversão na vantagem observada nos gráficos de desempenho. A plataforma AWS Fargate demonstrou ser a opção mais econômica, com um custo médio de $0,07 para a execução do teste, em contraste com os $0,09 da abordagem com EC2. Essa diferença de custo evidencia a principal proposta de valor do modelo de computação sem servidor: o Fargate fatura apenas pelos recursos de CPU e memória efetivamente consumidos durante a execução da tarefa, eliminando custos de capacidade ociosa. Em contrapartida, a instância EC2 incorre em custos pelo tempo total em

que permanece ativa, independentemente de estar processando requisições continuamente. Este resultado destaca o claro *trade-off* entre um desempenho ligeiramente superior e uma eficiência de custo significativamente maior, posicionando o Fargate como uma alternativa financeiramente mais vantajosa para este cenário de uso.

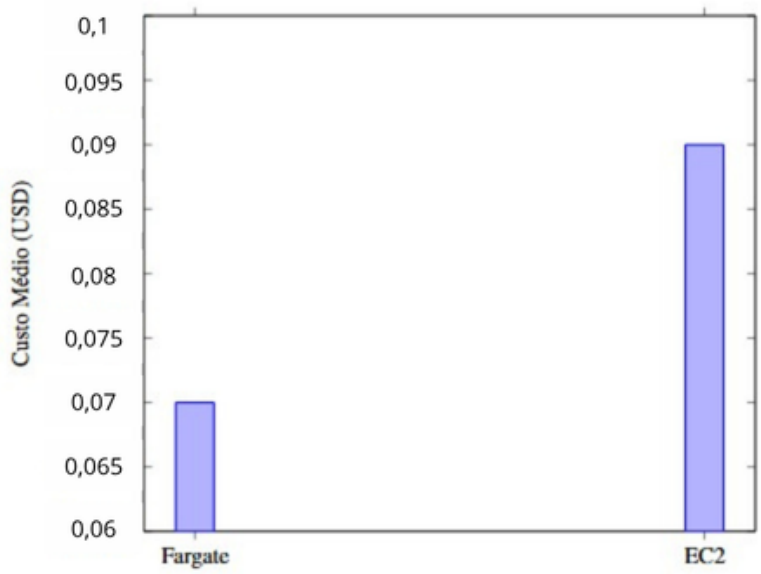
Figura 8 – Comparativo Tempo Médio por Imagem.



Fonte: Elaborado pelo Autor

O Gráfico da Figura 9 revela o aspecto financeiro da comparação, apresentando uma clara inversão na vantagem observada nos gráficos de desempenho. A plataforma AWS Fargate demonstrou ser a opção mais econômica, com um custo médio de $0,07 para a execução do teste, em contraste com os $0,09 da abordagem com EC2. Essa diferença de custo evidencia a principal proposta de valor do modelo de computação sem servidor: o Fargate fatura apenas pelos recursos de CPU e memória efetivamente consumidos durante a execução da tarefa, eliminando custos de capacidade ociosa. Em contrapartida, a instância EC2 incorre em custos pelo tempo total em que permanece ativa, independentemente de estar processando requisições continuamente. Este resultado destaca o claro *trade-off* entre um desempenho ligeiramente superior e uma eficiência de custo significativamente maior, posicionando o Fargate como uma alternativa financeiramente mais vantajosa para este cenário de uso.

Figura 9 – Comparativo Custo Médio.



Fonte: Elaborado pelo Autor

5.3 Considerações Finais da Seção

Um dos principais diferenciais entre as abordagens reside na simplicidade operacional, onde o AWS Fargate se destaca por sua menor complexidade de configuração em comparação com o ECS em instâncias EC2. Como uma plataforma sem servidor (*serverless*), o Fargate abstrai completamente a necessidade de provisionar, configurar ou gerenciar a infraestrutura subjacente; o desenvolvedor define apenas os recursos de CPU e memória para a tarefa, e a AWS se encarrega de executar e escalar a aplicação. Em contrapartida, a abordagem com EC2 exige um esforço operacional maior, pois o usuário é responsável por selecionar tipos de instância, configurar o sistema operacional, gerenciar a escalabilidade do *cluste*r e manter a segurança da infraestrutura. Essa simplicidade do Fargate reduz a sobrecarga operacional e permite que as equipes foquem na lógica da aplicação em vez de na gestão de servidores.

Em síntese, os experimentos demonstraram que, para o *offloading* computacional de classificação de imagem em dispositivos de baixa capacidade, a escolha entre AWS Fargate e AWS EC2 envolve um equilíbrio entre desempenho, custo e complexidade de gerenciamento. Embora o EC2 possa apresentar um tempo de processamento ligeiramente menor como pode-se observar nas Figuras(7 e 8), o Fargate oferece uma alternativa mais eficiente em termos de custo financeiro e simplicidade operacional observando a Figura (9). Esta validação reforça a aplicabilidade das plataformas de computação sem servidor em cenários de *offloading* de Aprendizado de Máquina, onde a automação e a escalabilidade são cruciais para otimizar

recursos e superar as limitações de hardware dos dispositivos clientes.

6 CONCLUSÕES E TRABALHOS FUTUROS

Este capítulo apresenta a síntese das descobertas e os resultados alcançados no presente trabalho. Inicialmente, serão retomados os objetivos propostos e discutido em que medida foram atingidos, consolidando as principais conclusões derivadas da análise comparativa das plataformas AWS Fargate e AWS EC2 no contexto do *offloading* computacional. Posteriormente, com base nos *insights* obtidos e nas limitações identificadas, serão sugeridas direções para futuras pesquisas, visando expandir o conhecimento e a aplicabilidade da solução proposta.

6.1 Conclusão

Este trabalho tem como objetivo principal avaliar a viabilidade de utilizar plataformas de execução de contêineres em nuvem para offloading computacional de aplicações de Aprendizado de Máquina em dispositivos de baixa capacidade computacional. Através da implementação de uma aplicação protótipo para classificação de imagem utilizando o modelo MobileNetV3Large e da condução de experimentos comparativos entre as abordagens AWS Fargate (computação sem servidor) e AWS EC2 (instâncias gerenciadas pelo usuário), é possível obter *insights* relevantes sobre desempenho e custo financeiro. É fundamental ressaltar, contudo, que estes resultados são específicos para a carga de trabalho e o modelo avaliados, não devendo ser generalizados para qualquer cenário de Aprendizado de Máquina sem investigações adicionais.

Os resultados experimentais demonstraram que:

- **Desempenho:** A abordagem AWS EC2 apresentou um tempo médio de inferência por imagem ligeiramente inferior (aproximadamente 2,89 segundos) e um tempo total de processamento marginalmente mais rápido (média de 1734,55 segundos) em comparação com o AWS Fargate (tempo médio de 3,03 segundos e tempo total de 1819,48 segundos). Essa diferença, embora pequena, pode ser atribuída ao maior controle sobre o ambiente e a alocação de recursos que o EC2 proporciona.
- **Custo Financeiro:** Em termos de custo financeiro, o AWS Fargate mostrou-se a opção mais econômica, com um custo médio de 0,07 USD por execução, enquanto o EC2 incorreu em 0,09 USD. Esta vantagem de custo do Fargate reflete seu modelo *pay-as-you-go*, que otimiza o gasto ao cobrar apenas pelos recursos efetivamente consumidos, alinhando-se à proposta da computação sem servidor de eliminar custos com capacidade ociosa.

- **Compromissos:** A escolha entre as plataformas reside em um compromisso entre um desempenho marginalmente superior (EC2) e a otimização de custo, simplicidade operacional e escalabilidade automática (Fargate). Para cenários de *offloading* computacional, onde a abstração da infraestrutura e a elasticidade são altamente desejáveis, o AWS Fargate surge como uma solução robusta e financeiramente eficiente, apesar de um pequeno aumento na latência percebida em comparação direta com o EC2.

Em suma, a pesquisa valida a aplicabilidade de plataformas de contêineres em nuvem para viabilizar o *offloading* computacional de cargas de trabalho intensivas de Aprendizado de Máquina de dispositivos com recursos limitados. O AWS Fargate, com sua natureza sem servidor, oferece uma alternativa convincente que simplifica a implantação, gerenciamento e escalabilidade de serviços de inferência de ML, tornando-os mais acessíveis e economicamente viáveis para uma ampla gama de aplicações que necessitam descarregar poder de processamento.

6.2 Trabalhos Futuros

Com base nas descobertas e limitações do presente estudo, sugere-se a continuidade da pesquisa em diversas direções:

- **Comparação com Soluções Locais (Baseline):** Integrar uma comparação dos resultados obtidos na nuvem com a execução local da mesma tarefa de classificação de imagem em diferentes dispositivos de baixa capacidade (ex: Raspberry Pi, smartphones). Isso quantificaria o real ganho do *offloading* em termos de desempenho e consumo energético no próprio dispositivo.
- **Otimização de Custos em EC2:** Investigar estratégias avançadas de otimização de custos para o AWS EC2, como o uso de *Spot Instances* ou *Reserved Instances*, e grupos de *Auto Scaling* mais granulares para aproximar o modelo de cobrança do *pay-as-you-go* do Fargate em cenários específicos.
- **Modelos de Offloading Dinâmico:** Explorar e implementar mecanismos de decisão de *offloading* mais inteligentes, onde o dispositivo decide dinamicamente se a inferência deve ser executada localmente ou na nuvem, com base em condições de rede, bateria e carga computacional do dispositivo.
- **Segurança e Sincronização:** Aprofundar a análise dos desafios do *offloading* relacionados à segurança da transferência de dados e aos mecanismos de sincronização para garantir a integridade e privacidade das informações em cenários de uso real.

- **Outros Modelos de Aprendizado de Máquina e Casos de Uso:** Ampliar o escopo dos experimentos para incluir outros modelos de Aprendizado de Máquina (ex: detecção de objetos, segmentação semântica) e diferentes tipos de *offloading* (ex: pré-processamento, pós-processamento, treinamento distribuído).
- **Avaliação de Latência de Rede:** Medir e analisar o impacto direto da latência da rede (localização do cliente vs. região da nuvem) na latência de ponta a ponta do *offloading*, considerando a variabilidade da conexão em ambientes reais.

Essas direções futuras podem enriquecer o campo de pesquisa em *offloading* computacional e Aprendizado de Máquina em ambientes de nuvem, contribuindo para soluções mais eficientes e adaptáveis para dispositivos com recursos limitados.

REFERÊNCIAS

Amazon Web Services. **Serverless Computing – Amazon Web Services**. 2024. Disponível em: https://aws.amazon.com/serverless/. Acesso em: 31 ago. 2024.

Amazon Web Services. **Amazon ECS on Amazon EC2**. 2025. Disponível em: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/launch_types.html. Acesso em: 28 jan. 2025.

Amazon Web Services. **Our Origins - AWS**. 2025. Disponível em: https://aws.amazon.com/pt/about-aws/our-origins/. Acesso em: 28 jan. 2025.

Amazon Web Services. **What is AWS Fargate?** 2025. Disponível em: https://aws.amazon.com/fargate/. Acesso em: 28 jan. 2025.

Awari. **Reconhecimento de Imagem em Machine Learning**: Conceitos e implementações. 2024. Disponível em: https://awari.com.br/reconhecimento-de-imagem-em-machine-learning-conceitos-e-implementacoes/. Acesso em: 31 ago. 2024.

BARANAUSKAS, J. A.; MONARD, M. C. **Reviewing some machine learning concepts and methods**. [*S. l.*], 2000.

BARBOSA, R. A.; ANDRADE, J. P. B.; SILVA, M. R. C. da; LIMA, F. I. d. S.; TRINTA, F. A. M.; REGO, P. A. L. Acelerando a identificação de impressões digitais através da técnica de offloading computacional. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES E SISTEMAS DISTRIBUÍDOS. **Anais**. [*S. l.*]: SBC, 2021. p. 29–42.

BARBOSA, R. A.; REGO, P. A. L. A mobile cloud computing testbed based on lightweight virtualization. In: IEEE. **2019 IEEE Global Communications Conference (GLOBECOM)**. [*S. l.*], 2019. p. 1–6.

BORGES, L. E. **Python para desenvolvedores**: aborda python 3.3. [*S. l.*]: Novatec Editora, 2014.

BRIKMAN, Y. **Terraform: up and running**: writing infrastructure as code. [*S. l.*]: O'Reilly Media, Inc., 2022.

BUYYA, R.; YEO, C. S.; VENUGOPAL, S.; BROBERG, J.; BRANDIC, I. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. **Future Generation Computer Systems**, Elsevier, v. 25, n. 6, p. 599–616, 2009.

CARR, N. **The big switch**: Rewiring the world, from edison to google. [*S. l.*]: WW Norton & Company, 2009.

CHEN, G. **Modernizing Applications with Containers in Public Cloud**. [*S. l.*], 2019.

COSTA, G. A.; COUTO, D. P. **Inteligência Artificial Aplicada ao Reconhecimento de Imagens**. 2008. Trabalho de Conclusão de Curso II – PMR2550, Escola Politécnica da Universidade de São Paulo, Orientador: Prof. Fábio G. Cozman.

DIO - Digital Innovation One. **O que você precisa saber antes de aprender Docker**. 2025. Disponível em: https://www.dio.me/articles/o-que-voce-precisa-saber-antes-de-aprender-docker. Acesso em: 28 jan. 2025.

ELYAN, E.; VUTTIPITTAYAMONGKOL, P.; JOHNSTON, P.; MARTIN, K.; MCPHERSON, K.; JAYNE, C.; SARKER, M. K. *et al.* Computer vision and machine learning for medical image analysis: recent advances, challenges, and way forward. **Artificial Intelligence Surgery**, OAE Publishing, v. 2, 2022.

EYK, E. van; IOSUP, A.; ABAD, C. L.; GROHMANN, J.; EISMANN, S. A spec rg cloud group's vision on faas: The serverless computing manifesto. In: IEEE. **2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)**. S.l., 2018. p. 165–170.

FLORES, H.; HUI, P.; TARKOMA, S.; LI, Y.; SRIRAMA, S.; BUYYA, R. Mobile code offloading: from concept to practice and beyond. **IEEE Communications Magazine**, v. 53, n. 3, p. 80–88, 2015.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. MIT Press, 2016. Disponível em: http://www.deeplearningbook.org.

GRINBERG, M. **Flask web development**. [*S. l.*]: O'Reilly Media, Inc., 2018.

HOWARD, A.; SANDLER, M.; CHU, G.; CHEN, L.-C.; CHEN, B.; TAN, M.; WANG, W.; ZHU, Y.; PANG, R.; VASUDEVAN, V. *et al.* Searching for mobilenetv3. In: IEEE/CVF INTERNATIONAL CONFERENCE ON COMPUTER VISION. **Proceedings**. [*S. l.*], 2019. p. 1314–1324.

IBM. **O que é Machine Learning?** 2024. Disponível em: https://www.ibm.com/br-pt/topics/machine-learning. Acesso em: 31 ago. 2024.

JAIN, R. **The Art of Computer Systems Performance Analysis**: Techniques for experimental design, measurement, simulation, and modeling. [*S. l.*]: Wiley Professional Computing, 1991.

JAYARAM, V.; SANKITI, S. R.; KRISHNAPPA, M. S.; VEERAPANENI, P. K.; CARIMIREDDY, P. K. Accelerated cloud infrastructure development using terraform. **International Journal of Emerging Technologies and Innovative Research**, v. 11, n. 9, p. f382–f387, 2024.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. **Advances in neural information processing systems**, v. 25, 2012.

LI, Y. **Deep Reinforcement Learning**: An overview. 2018. Disponível em: https://arxiv.org/abs/1701.07274. Acesso em: 31 ago. 2024.

LUDERMIR, T. B. Inteligência artificial e aprendizado de máquina: estado atual e tendências. **Estudos Avançados**, SciELO Brasil, v. 35, p. 85–94, 2021.

MELL, P.; GRANCE, T. **The NIST definition of cloud computing**. [*S. l.*], 2011.

Microsoft. **Discutir diferentes tipos de modelos de nuvem**. 2024. Disponível em: https://azure.microsoft.com/pt-br/resources/cloud-computing-dictionary/types-of-cloud-computing. Acesso em: 2024-08-27.

MOREIRA, R. d. S.; COUTINHO, E. F.; ALENCAR, J. M. U. Avaliação das linguagens python e go na plataforma aws lambda para microsserviços de computação de alto desempenho. In: **WSCAD 2021**. Quixadá, CE, Brasil: Universidade Federal do Ceará (UFC), 2021.

PANG, B.; NIJKAMP, E.; WU, Y. N. Deep learning with tensorflow: A review. **Journal of Educational and Behavioral Statistics**, SAGE Publications Sage CA: Los Angeles, CA, v. 45, n. 2, p. 227–248, 2020.

RAHMAN, M. **Serverless cloud computing**: a comparative analysis of performance, cost, and developer experiences in container-level services. Dissertação (Mestrado) – University of Stavanger, Noruega, 2023.

RAHMAN, M. **Serverless cloud computing**: a comparative analysis of performance, cost, and developer experiences in container-level services. Dissertação (Mestrado) – University of Stavanger, Noruega, 2023.

Red Hat. **O que é Docker?** 2025. Disponível em: https://www.redhat.com/pt-br/topics/containers/what-is-docker. Acesso em: 28 jan. 2025.

RUSSAKOVSKY, O.; DENG, J.; SU, H.; KRAUSE, J.; SATHEESH, S.; MA, S.; HUANG, Z.; KARPATHY, A.; KHOSLA, A.; BERNSTEIN, M.; BERG, A. C.; LI, F.-F. Imagenet large scale visual recognition challenge. **International Journal of Computer Vision**, Springer, v. 115, n. 3, p. 211–252, 2015.

SANKARANARAYANAN, S.; KAVITHA, C. S. A comprehensive survey on image classification techniques. In: NAVEEN, J.; PRATHOSH, A.; SANYAL, S.; PRATHYUSHA, K. (Ed.). **Intelligent Computing and Communication**. Singapore: Springer, 2020. p. 25–34.

SANTOS, B. P.; SILVA, L. A.; CELES, C. S. F. S.; BORGES, J. B.; NETO, B. S. P.; VIEIRA, M. A. M.; VIEIRA, L. F. M.; GOUSSEVSKAIA, O. N.; LOUREIRO, A. Internet das coisas: da teoria à prática. In: MINICURSOS DO SIMPóSIO BRASILEIRO DE REDES DE COMPUTADORES E SISTEMAS DISTRIBUíDOS. **Anais**. S.l.: SBC, 2016. p. 16.

SILVA, H. B. d. **Uma investigação sobre o processo migratório para a plataforma de computação em nuvem no Brasil**. Dissertação (Mestrado) – Universidade Federal de Pernambuco, 2016.

SILVA, V. J. d.; BONACELLI, M. B. M.; PACHECO, C. A. O sistema tecnológico digital: inteligência artificial, computação em nuvem e big data. **Revista Brasileira de Inovação**, SciELO Brasil, v. 19, p. e0200024, 2020.

TIHFON, G. M.; PARK, S.; KIM, J.; KIM, Y.-M. An efficient multi-task paas cloud infrastructure based on docker and aws ecs for application deployment. **Cluster Computing**, Springer, v. 19, p. 1585–1597, 2016.

YAO, S.; LI, J.; LIU, D.; WANG, T.; LIU, S.; SHAO, H.; ABDELZAHER, T. Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency. In: CONFERENCE ON EMBEDDED NETWORKED SENSOR SYSTEMS, 18. **Proceedings**. [*S. l.*], 2020. p. 476–488.